SysV Command Reference

005798-A00

apollo

SysV Command Reference

Order No. 005798-A00

Apollo Computer Inc.
330 Billerica Road
Chelmsford, MA 01824

First Printing: July 1988

Preface

The *SysV Command Reference* describes the user commands and games available in Domain®/OS SysV. This manual is intended for users who are familiar with System V Release 3 UNIX software and Domain/OS. It provides neither a general overview of Domain/OS SysV nor details of the implementation of the system. We assume that you are already familiar with the material in *Using Your SysV Environment.*

We have divided the *SysV Command Reference* into two sections:

Commands	Section 1 provides reference material on user commands.
Games	Section 6 provides reference material on games.

Each section consists of independent entries of a page or so each. The name of the entry is in the upper corners of its pages, together with the section number, and sometimes a letter characteristic of a class. For example, the **ftp** command is 1C. Each section begins with **intro**(*N*), followed by **domain**(*N*), where *N* is the number of the section. Entries thereafter appear in alphabetical order.

Some entries may describe several features. In such cases, the entry may appear only once, alphabetized under its ''primary'' name, the name that appears at the upper corners of each manual page.

Entries with Domain/OS SysV (as contrasted with a simple SysV) centered at the top of the page describe features unique to Domain/OS SysV. Each section contains an entry with the name **domain** that provides an overview of the unique features in that section.

We use the convention **name**(*N*) to cite entry **name** in section *N* of this and other manuals. References to sections other than 1 or 6 mean that **name** is contained in another manual. The *SysV Programmer's Reference* contains Sections 2, 3, 4, and 5. *Managing SysV System Software* includes Sections 1M and 7.

All entries are based on a common format, not all of whose parts always appear:

NAME	Gives the name of the feature and briefly states its purpose.
SYNOPSIS	Summarizes the use of the feature being described. In the case of system calls and subroutines, this summary usually specifies header files (by way of the appropriate **#include** <*file.h*> preprocessor statement) containing definitions needed by the call or subroutine. This summary also usually contains a set of declarations as they might appear in a C-language function header defining the call or subroutine.
DESCRIPTION	Describes the feature.
EXAMPLE(S)	Gives example(s) of usage, where appropriate.
FILES	Gives the filenames that are built into the feature.
SEE ALSO	Gives pointers to related information.
DIAGNOSTICS	Discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.
NOTES	Gives generally ''helpful hints'' about the use of the feature.
WARNINGS	Points out potential pitfalls.
BUGS	Gives known bugs and sometimes deficiencies.
CAVEATS	Gives details of the implementation that might affect usage.

A ''Table of Contents'' and a ''Permuted Index'' derived from that table precede Section 1. The Permuted Index is a list of keywords, given in the second of three columns, together with the context in which each keyword is found. Keywords are either topical keywords or the names of manual entries. Entries are identified with their section numbers shown in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from components that exist only to exercise a particular system call. The right column lists the name of the manual page on which each keyword may be found. The left column contains useful information about the keyword.

Online Access

We deliver a machine-readable version of this manual (and Sections 1M and 7) in the files

/sys5.3/usr/catman/u_man/man[16]/*name***.[16]***class*,

and

/sys5.3/usr/catman/a_man/man[17]/*name***.[1M7]***class*, where *name* is that of the feature documented, **[1671M]** is either 1, 6, 7, or 1M depending upon the section, and *class* (C for communication, G for graphics, etc.) may or may not appear.

If you have installed these files on your workstation, or you have links from your workstation to one where these files are installed, you may access them by way of the **man**(1) command. (To read about **man**, type

$ man 1 man

or refer to **man**(1) in this book).

Related Manuals

The file **/install/doc/apollo/os.v.** "latest software release number" *__manuals* lists current titles and revisions for all available manuals.

For example, at Software Release 10 (SR10.0) refer to the file **/install/doc/apollo/os.v.10.0__manuals** to check that you are using the correct version of manuals. You may also want to use this file to check that you have ordered all of the manuals that you need.

(If you are using the (Aegis™, environment, you can access the same information through the Help system by typing **help manuals**.)

Refer to the *Domain Documentation Quick Reference* (002685) and the *Domain Documentation Master Index* (011242) for a complete list of related documents.

For introductory information about the Domain/OS system and details about using the SysV environment, refer to the following documents:

• *Getting Started with Domain/OS*	(002348)
• *Using Your SysV Environment*	(011022)
• *Domain Display Manager Command Reference*	(011418)

For more information on programming in the Domain/OS SysV environment, refer to the following documents:

• *Domain/OS Call Reference*, Volumes 1 and 2	(007196 and 012888)
• *Domain/OS Programming Environment Reference*	(011010)
• *Domain Binder and Librarian Reference*	(004977)
• *Domain C Language Reference*	(002093)

• *SysV Programmer's Reference*	(005799)
• *Managing SysV System Software*	(010851)

Problems, Questions, and Suggestions

We appreciate comments from the people who use our system. To make it easy for you to communicate with us, we provide the Apollo® Product Reporting (APR) system for comments related to hardware, software, and documentation. By using this formal channel you make it easy for us to respond to your comments.

You can get more information about how to submit an APR by consulting the appropriate Command Reference manual for your environment (Aegis, BSD, or SysV). Refer to the **mkapr** shell command description. You can view the same description online by typing:

$ man 1 mkapr (in the SysV environment)

% **man 1 mkapr** (in the BSD environment)

$ help mkapr (in the Aegis environment)

Alternatively, you may use the Reader's Response form at the back of this manual to submit comments about the manual.

Documentation Conventions

This manual uses the following symbolic conventions:

literal values	Bold words or characters in formats and command descriptions represent commands or keywords that you must use literally. Bold words in text indicate the first use of a new term. Filenames and pathnames are also in bold.
user-supplied values	Placeholders for symbols that you must supply are printed in italics. For example, the names chosen for call arguments appear in italics.
sample user input	In samples, information that the user enters appears in bold.
`examples`	Examples of program code appear in `this typeface`.

[ ]	Square brackets enclose optional items in formats and command descriptions.
{ }	Braces enclose a list from which you must choose an item in formats and command descriptions.
|	A vertical bar separates items in a list of choices.
...	Ellipses mean that the previous argument-prototype may be repeated.
–	An argument beginning with a minus sign (''–'') usually means that it is an option-specifying argument used by the command itself, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with ''–''.
——— ⊞ ———	This symbol indicates the end of a section.

Contents

SysV Commands

functions of DASI 300 and/	300, 300s: handle special	300(1)
special functions of DASI	300 and 300s terminals/handle	300(1)
of DASI 300 and 300s/ 300,	300s: handle special functions	300(1)
functions of DASI 300 and	300s terminals/handle special	300(1)
comparison diff3:	3-way differential file	diff3(1)
of the DASI 450 terminal	450: handle special functions	450(1)
special functions of the DASI	450 terminal450: handle	450(1)
f77: Fortran	77 compiler	f77(1)
of a file touch: update	access and modification times	touch(1)
chacl: change	access control list	chacl(1)
cpacl: copy	access control list	cpacl(1)
lsacl: list	access control list	lsacl(1)
dbacl: Domain/Dialog based	access control list editor	dbacl(1)
/unstr: create a random	access file for storing/	strfile(6)
commands graphics:	access graphic and numeric	graphics(1G)
collected by trace modules on	active Streams/print messages	trmon(1)
STREAMS trace/ trconf: list	active Streams or configure	trconf(1)
current SCCS file editing	activitysact: print	sact(1)
report process data and system	activity/time a command;	timex(1)
random, hopefully interesting,	adagefortune: print a	fortune(6)
esa: display	address of external symbol	esa(1)
list objects mapped into the	address spacelas:	las(1)
SCCS files	admin: create and administer	admin(1)
admin: create and	administer SCCS files	admin(1)
battlestar: a tropical	adventure game	battlestar(6)
file for printing by Domain/OS	Aegis print spooler/queue a	prf(1)
sort: sort	and/or merge files	sort(1)
terminal worms:	animate worms on a display	worms(6)
rain:	animated raindrops display	rain(6)
bcd: convert to	antique media	bcd(6)
mkapr: make an	Apollo product report	mkapr(1)
maintainer for portable/	ar: archive and library	ar(1)
number: convert	Arabic numerals to English	number(6)
language bc:	arbitrary-precision arithmetic	bc(1)
for portable archives ar:	archive and library maintainer	ar(1)
convert lbr libraries to SR10	archive librarieslbr2ar:	lbr2ar(1)
tar: tape file	archiver	tar(1)
maintainer for portable	archives/archive and library	ar(1)
cpio: copy file	archives in and out	cpio(1)
command xargs: construct	argument list(s) and execute	xargs(1)
echo: echo	arguments	echo(1)
expr: evaluate	arguments as an expression	expr(1)
bc: arbitrary-precision	arithmetic language	bc(1)
ftp:	ARPANET file transfer program	ftp(1C)
expr: evaluate arguments	as an expression	expr(1)
characters asa: interpret	ASA carriage control	asa(1)
control characters	asa: interpret ASA carriage	asa(1)
help:	ask for SCCS help	help(1)
a later time	at, batch: execute commands at	at(1)
a later time	at, batch: execute commands at	batch(1)
scrattr: screen	attributes	scrattr(1)
wait:	await completion of process	wait(1)
processing language	awk: pattern scanning and	awk(1)
backgammon: the game of	backgammon	backgammon(6

btlgammon: the game of	backgammon	btlgammon(6)
teachgammon: teach the game of	backgammon	teachgammon(6)
backgammon	backgammon: the game of	backgammon(6)
bgcolor: make interesting	background colors	bgcolor(6)
or index a magnetic media	backup filerbak: restore	rbak(1)
wbak: create a magnetic media	backup file	wbak(1)
	banner: make posters	banner(1)
banner: print large	banner on printer	banner(6)
printer	banner: print large banner on	banner(6)
editor dbacl: Domain/Dialog	based access control list	dbacl(1)
(visual) display editor	based on ex/screen-oriented	vi(1)
portions of path names	basename, dirname: deliver	basename(1)
portions of path names	basename, dirname: deliver	dirname(1)
later time at,	batch: execute commands at a	at(1)
later time at,	batch: execute commands at a	batch(1)
adventure game	battlestar: a tropical	battlestar(6)
arithmetic language	bc: arbitrary-precision	bc(1)
	bcd: convert to antique media	bcd(6)
	bdiff: big diff	bdiff(1)
cb: C program	beautifier	cb(1)
	bfs: big file scanner	bfs(1)
background colors	bgcolor: make interesting	bgcolor(6)
via mail /encode/decode a	binary file for transmission	uudecode(1C)
via mail /encode/decode a	binary file for transmission	uuencode(1C)
a type object module for	bindingcrtyobj: create	crtyobj(1)
	bj: the game of blackjack	bj(6)
bj: the game of	blackjack	bj(6)
system was built	bldt: display time operating	bldt(1)
sum: print checksum and	block count of a file	sum(1)
boggle: play the game of	boggle	boggle(6)
boggle	boggle: play the game of	boggle(6)
rsh: the standard/restricted	Bourne Shell (command/sh,	rsh(1)
rsh: the standard/restricted	Bourne Shell (command/sh,	sh(1)
mille: play Mille	Bournes	mille(6)
comment	btlfortune: print a random	btlfortune(6)
backgammon	btlgammon: the game of	btlgammon(6)
	btlhangman: guess the word	btlhangman(6)
time operating system was	builtbldt: display	bldt(1)
size: print section sizes in	bytes of common object files	size(1)
cc:	C compiler	cc(1)
cflow: generate	C flowgraph	cflow(1)
cpp: the	C language preprocessor	cpp(1)
cb:	C program beautifier	cb(1)
lint: a	C program checker	lint(1)
cxref: generate	C program cross-reference	cxref(1)
ctrace:	C program debugger	ctrace(1)
object file list: produce	C source listing from a common	list(1)
	cal: print calendar	cal(1)
dc: desk	calculator	dc(1)
cal: print	calendar	cal(1)
	calendar: reminder service	calendar(1)
cu:	call another UNIX system	cu(1C)
to an LP line printer lp,	cancel: send/cancel requests	cancel(1)
to an LP line printer lp,	cancel: send/cancel requests	lp(1)
the solitaire card game	canfieldcanfield, cfscores:	canfield(6)
solitaire card game canfield	canfield, cfscores: the	canfield(6)

iso:	convert files to ISO format	swedish_to_iso(1)
iso:	convert files to ISO format	swiss_to_iso(1)
iso:	convert files to ISO format	uk_to_iso(1)
SR10 format cvt_font:	convert fonts from pre-SR10 to	cvt_font(1)
archive libraries lbr2ar:	convert lbr libraries to SR10	lbr2ar(1)
COFF format modules obj2coff:	convert OBJ format modules to	obj2coff(1)
upper and lowercase/ cvtname:	convert pathnames between	cvtname(1)
and SR10 formats cvtrgy:	convert registry between SR9.x	cvtrgy(1)
bcd:	convert to antique media	bcd(6)
rcp: remote file	copy	rcp(1C)
uuname: UNIX-to-UNIX system	copyuucp, uulog,	uucp(1C)
uuname: UNIX-to-UNIX system	copyuucp, uulog,	uulog(1C)
uuname: UNIX-to-UNIX system	copyuucp, uulog,	uuname(1C)
UNIX-to-UNIX system file	copyuuto, uupick: public	uupick(1C)
UNIX-to-UNIX system file	copyuuto, uupick: public	uuto(1C)
dd: convert and	copy a file	dd(1)
cpacl:	copy access control list	cpacl(1)
cpio:	copy file archives in and out	cpio(1)
cp:	copy files	cp(1)
file cpscr:	copy the current display to a	cpscr(1)
wc: word	count	wc(1)
sum: print checksum and block	count of a file	sum(1)
hpc: program	counter histogram	hpc(1)
	cp: copy files	cp(1)
list	cpacl: copy access control	cpacl(1)
and out	cpio: copy file archives in	cpio(1)
preprocessor	cpp: the C language	cpp(1)
display to a file	cpscr: copy the current	cpscr(1)
craps: the game of	craps	craps(6)
	craps: the game of craps	craps(6)
modify a device descriptor/	crddf: create, display, or	crddf(1)
ln:	create a hard or soft link	ln(1)
file wbak:	create a magnetic media backup	wbak(1)
crty:	create a new type	crty(1)
node crp:	create a process on a remote	crp(1)
for storing/ strfile, unstr:	create a random access file	strfile(6)
window crpad:	create a transcript pad and	crpad(1)
for binding crtyobj:	create a type object module	crtyobj(1)
files admin:	create and administer SCCS	admin(1)
device descriptor file crddf:	create, display, or modify a	crddf(1)
STREAMS drivers and/ mksinit:	create initialization code for	mksinit(1)
cribbage: the card game	cribbage	cribbage(6)
cribbage	cribbage: the card game	cribbage(6)
crontab: user	crontab file	crontab(1)
	crontab: user crontab file	crontab(1)
cxref: generate C program	cross-reference	cxref(1)
remote node	crp: create a process on a	crp(1)
and window	crpad: create a transcript pad	crpad(1)
pg: file perusal filter for	CRTs	pg(1)
	crty: create a new type	crty(1)
module for binding	crtyobj: create a type object	crtyobj(1)
	csplit: context split	csplit(1)
	ctrace: C program debugger	ctrace(1)
	cu: call another UNIX system	cu(1C)
cpscr: copy the	current display to a file	cpscr(1)
set or print identifier of	current host systemhostid:	hostid(1)

hostname: set or print name of	current host system	hostname(1)
activity sact: print	current SCCS file editing	sact(1)
uname: print name of	current UNIX system	uname(1)
spline: interpolate smooth	curve	spline(1G)
of each line of a file	cut: cut out selected fields	cut(1)
each line of a file cut:	cut out selected fields of	cut(1)
pre-SR10 to SR10 format	cvt_font: convert fonts from	cvt_font(1)
between upper and lowercase/	cvtname: convert pathnames	cvtname(1)
between SR9.x and SR10/	cvtrgy: convert registry	cvtrgy(1)
cross-reference	cxref: generate C program	cxref(1)
flake: induce terminal	dandruff	flake(6)
directory service whois:	DARPA Internet username	whois(1)
/handle special functions of	DASI 300 and 300s terminals	300(1)
special functions of the	DASI 450 terminal450: handle	450(1)
prof: display profile	data	prof(1)
/time a command; report process	data and system activity	timex(1)
a terminal or query terminfo	databasetput: initialize	tput(1)
join: relational	database operator	join(1)
date: print and set the	date	date(1)
	date: print and set the date	date(1)
access control list editor	dbacl: Domain/Dialog based	dbacl(1)
	dbx: debugger	dbx(1)
	dc: desk calculator	dc(1)
	dd: convert and copy a file	dd(1)
Debugging Environment	dde: Domain Distributed	dde(1)
ctrace: C program	debugger	ctrace(1)
dbx:	debugger	dbx(1)
dde: Domain Distributed	Debugging Environment	dde(1)
dlty:	delete a type	dlty
basename, dirname:	deliver portions of path names	basename(1)
basename, dirname:	deliver portions of path names	dirname(1)
file tail:	deliver the last part of a	tail(1)
delta commentary of an SCCS	deltacdc: change the	cdc(1)
delta: make a	delta (change) to an SCCS file	delta(1)
delta cdc: change the	delta commentary of an SCCS	cdc(1)
rmdel: remove a	delta from an SCCS file	rmdel(1)
to an SCCS file	delta: make a delta (change)	delta(1)
comb: combine SCCS	deltas	comb(1)
mesg: permit or	deny messages	mesg(1)
display, or modify a device	descriptor file/create,	crddf(1)
edmtdesc: edit magtape	descriptor file	edmtdesc(1)
dc:	desk calculator	dc(1)
file:	determine file type	file(1)
create, display, or modify a	device descriptor filecrddf:	crddf(1)
/tekset, td: graphical	device routines and filters	erase(1G)
/tekset, td: graphical	device routines and filters	gdev(1G)
/tekset, td: graphical	device routines and filters	hardcopy(1G)
/tekset, td: graphical	device routines and filters	hpd(1G)
/tekset, td: graphical	device routines and filters	td(1G)
/tekset, td: graphical	device routines and filters	tekset(1G)
ratfor: rational FORTRAN	dialect	ratfor(1)
bdiff: big	diff	bdiff(1)
comparator	diff: differential file	diff(1)
comparison	diff3: 3-way differential file	diff3(1)
sdiff: side-by-side	difference program	sdiff(1)
diff:	differential file comparator	diff(1)

	ed, red: text editor	ed(1)
	ed, red: text editor	red(1)
	edfont: edit a character font	edfont(1)
edfont:	edit a character font	edfont(1)
edmtdesc:	edit magtape descriptor file	edmtdesc(1)
ex for casual users)	edit: text editor (variant of	edit(1)
sact: print current SCCS file	editing activity	sact(1)
based access control list	editordbacl: Domain/Dialog	dbacl(1)
ed, red: text	editor	ed(1)
ex: text	editor	ex(1)
ed, red: text	editor	red(1)
sed: stream	editor	sed(1)
/(visual) display	editor based on ex	vi(1)
ld: link	editor for common object files	ld(1)
casual users) edit: text	editor (variant of ex for	edit(1)
descriptor file	edmtdesc: edit magtape	edmtdesc(1)
pattern using full regular/	egrep: search a file for a	egrep(1)
	emt: emulate a dumb terminal	emt(1)
emt:	emulate a dumb terminal	emt(1)
vt100: VT100 terminal	emulator	vt100(1)
enable/disable LP printers	enable, disable:	disable(1)
enable/disable LP printers	enable, disable:	enable(1)
enable, disable:	enable/disable LP printers	disable(1)
enable, disable:	enable/disable LP printers	enable(1)
for/ uuencode,uudecode:	encode/decode a binary file	uudecode(1C)
for/ uuencode,uudecode:	encode/decode a binary file	uuencode(1C)
sccs: front	end for the SCCS subsystem	sccs(1)
convert Arabic numerals to	Englishnumber:	number(6)
logger: make	entries in the system log	logger(1)
man: print	entries in this manual	man(1)
command execution	env: set environment for	env(1)
Domain Distributed Debugging	Environmentdde:	dde(1)
execution env: set	environment for command	env(1)
graphical device/ gdev: hpd,	erase, hardcopy, tekset, td:	erase(1G)
graphical device/ gdev: hpd,	erase, hardcopy, tekset, td:	gdev(1G)
graphical device/ gdev: hpd,	erase, hardcopy, tekset, td:	hardcopy(1G)
graphical device/ gdev: hpd,	erase, hardcopy, tekset, td:	hpd(1G)
graphical device/ gdev: hpd,	erase, hardcopy, tekset, td:	td(1G)
graphical device/ gdev: hpd,	erase, hardcopy, tekset, td:	tekset(1G)
hashcheck: find spelling	errors/hashmake, spellin,	hashcheck(1)
hashcheck: find spelling	errors/hashmake, spellin,	hashmake(1)
hashcheck: find spelling	errors/hashmake, spellin,	spell(1)
hashcheck: find spelling	errors/hashmake, spellin,	spellin(1)
external symbol	esa: display address of	esa(1)
expression expr:	evaluate arguments as an	expr(1)
test: condition	evaluation command	test(1)
display editor based on	ex/screen-oriented (visual)	vi(1)
edit: text editor (variant of	ex for casual users)	edit(1)
	ex: text editor	ex(1)
shell xdmc:	execute a DM command from the	xdmc(1)
construct argument list(s) and	execute commandxargs:	xargs(1)
time at, batch:	execute commands at a later	at(1)
time at, batch:	execute commands at a later	batch(1)
set environment for command	executionenv:	env(1)
UNIX-to-UNIX system command	executionuux:	uux(1C)
sleep: suspend	execution for an interval	sleep(1)

pcat, unpack: compress and	expand filespack,	pack(1)
pcat, unpack: compress and	expand filespack,	pcat(1)
pcat, unpack: compress and	expand filespack,	unpack(1)
expression	expr: evaluate arguments as an	expr(1)
expr: evaluate arguments as an	expression	expr(1)
regcmp: regular	expression compile	regcmp(1)
a pattern using full regular	expressions/search a file for	egrep(1)
commands and	extensions/Domain/OS-specific	domain(1)
esa: display address of	external symbol	esa(1)
cat	f1 concatenate and print files	cat(1)
	f77: Fortran 77 compiler	f77(1)
	factor: factoring program	factor(6)
factors of a number	factor: obtain the prime	factor(1)
factor:	factoring program	factor(6)
factor: obtain the prime	factors of a number	factor(1)
true,	false: provide truth values	false(1)
true,	false: provide truth values	true(1)
fst: print	fault status information	fst(1)
col: filter reverse line	feeds	col(1)
character string	fgrep: search a file for a	fgrep(1)
robots:	fight off villainous robots	robots(6)
copy the current display to a	filecpscr:	cpscr(1)
or modify a device descriptor	filecrddf: create, display,	crddf(1)
crontab: user crontab	file	crontab(1)
fields of each line of a	filecut: cut out selected	cut(1)
dd: convert and copy a	file	dd(1)
a delta (change) to an SCCS	filedelta: make	delta(1)
selected parts of an object	filedump: dump	dump(1)
edit magtape descriptor	fileedmtdesc:	edmtdesc(1)
get: get a version of an SCCS	file	get(1)
listing from a common object	filelist: produce C source	list(1)
change the format of a text	filenewform:	newform(1)
name list of common object	filenm: print	nm(1)
or subsequent lines of one	file/lines of several files	paste(1)
prs: print an SCCS	file	prs(1)
index a magnetic media backup	filerbak: restore or	rbak(1)
remove a delta from an SCCS	filermdel:	rmdel(1)
two versions of an SCCS	filesccsdiff: compare	sccsdiff(1)
from a common object	file/line number information	strip(1)
checksum and block count of a	filesum: print	sum(1)
deliver the last part of a	filetail:	tail(1)
and modification times of a	filetouch: update access	touch(1)
undo a previous get of an SCCS	fileunget:	unget(1)
report repeated lines in a	fileuniq:	uniq(1)
val: validate SCCS	file	val(1)
create a magnetic media backup	filewbak:	wbak(1)
tar: tape	file archiver	tar(1)
cpio: copy	file archives in and out	cpio(1)
mcs: manipulate the object	file comment section	mcs(1)
diff: differential	file comparator	diff(1)
diff3: 3-way differential	file comparison	diff3(1)
rcp: remote	file copy	rcp(1C)
public UNIX-to-UNIX system	file copyuuto, uupick:	uupick(1C)
public UNIX-to-UNIX system	file copyuuto, uupick:	uuto(1C)
	file: determine file type	file(1)
sact: print current SCCS	file editing activity	sact(1)

fgrep: search a	file for a character string	fgrep(1)
grep: search a	file for a pattern	grep(1)
regular/ egrep: search a	file for a pattern using full	egrep(1)
Aegis print/ prf: queue a	file for printing by Domain/OS	prf(1)
/unstr: create a random access	file for storing strings	strfile(6)
/encode/decode a binary	file for transmission via mail	uudecode(1C)
/encode/decode a binary	file for transmission via mail	uuencode(1C)
chsh, passwd: change password	file informationchfn,	chfn(1)
chsh, passwd: change password	file informationchfn,	chsh(1)
chsh, passwd: change password	file informationchfn,	passwd(1)
split: split a	file into pieces	split(1)
pg:	file perusal filter for CRTs	pg(1)
bfs: big	file scanner	bfs(1)
ftp: ARPANET	file transfer program	ftp(1C)
tftp: trivial	file transfer protocol	tftp(1C)
file: determine	file type	file(1)
umask: set	file-creation mode mask	umask(1)
create and administer SCCS	filesadmin:	admin(1)
cat f1 concatenate and print	files	cat(1)
cmp: compare two	files	cmp(1)
lines common to two sorted	filescomm - select or reject	comm(1)
cp: copy	files	cp(1)
find: find	files	find(1)
split FORTRAN or ratfor	filesfsplit:	fsplit(1)
link editor for common object	filesld:	ld(1)
mv: move	files	mv(1)
unpack: compress and expand	filespack, pcat,	pack(1)
unpack: compress and expand	filespack, pcat,	pcat(1)
pr: print	files	pr(1)
in bytes of common object	files/print section sizes	size(1)
sort: sort and/or merge	files	sort(1)
unpack: compress and expand	filespack, pcat,	unpack(1)
what: identify SCCS	files	what(1)
rm, rmdir: remove	files or directories	rm(1)
rm, rmdir: remove	files or directories	rmdir(1)
/merge same lines of several	files or subsequent lines of/	paste(1)
iso: convert	files to ISO format	french_to_iso(1)
iso: convert	files to ISO format	german_to_iso(1)
iso: convert	files to ISO format	iso(1)
iso: convert	files to ISO format	nor.dan_to_iso(1)
iso: convert	files to ISO format	swedish_to_iso(1)
iso: convert	files to ISO format	swiss_to_iso(1)
iso: convert	files to ISO format	uk_to_iso(1)
greek: select terminal	filter	greek(1)
nl: line numbering	filter	nl(1)
pg: file perusal	filter for CRTs	pg(1)
col:	filter reverse line feeds	col(1)
graphical device routines and	filters/hardcopy, tekset, td:	erase(1G)
graphical device routines and	filters/hardcopy, tekset, td:	gdev(1G)
graphical device routines and	filters/hardcopy, tekset, td:	hardcopy(1G)
graphical device routines and	filters/hardcopy, tekset, td:	hpd(1G)
graphical device routines and	filters/hardcopy, tekset, td:	td(1G)
graphical device routines and	filters/hardcopy, tekset, td:	tekset(1G)
tplot: graphics	filters	tplot(1G)
find:	find files	find(1)
	find: find files	find(1)

backgammon: the	game of backgammon	backgammon(6)
btlgammon: the	game of backgammon	btlgammon(6)
teachgammon: teach the	game of backgammon	teachgammon(6)
bj: the	game of blackjack	bj(6)
boggle: play the	game of boggle	boggle(6)
craps: the	game of craps	craps(6)
wump: the	game of hunt-the-wumpus	wump(6)
domain: Domain/OS-specific	games	domain(6)
intro: introduction to	games	intro(6)
tekset, td: graphical device/	gdev: hpd, erase, hardcopy,	erase(1G)
tekset, td: graphical device/	gdev: hpd, erase, hardcopy,	gdev(1G)
tekset, td: graphical device/	gdev: hpd, erase, hardcopy,	hardcopy(1G)
tekset, td: graphical device/	gdev: hpd, erase, hardcopy,	hpd(1G)
tekset, td: graphical device/	gdev: hpd, erase, hardcopy,	td(1G)
tekset, td: graphical device/	gdev: hpd, erase, hardcopy,	tekset(1G)
maze:	generate a maze	maze(6)
cflow:	generate C flowgraph	cflow(1)
cross-reference cxref:	generate C program	cxref(1)
lexical tasks lex:	generate programs for simple	lex(1)
Domain/Dialogue-based moire	generatordmoire:	dmoire(6)
random: random number	generator	random(6)
get:	get a version of an SCCS file	get(1)
file	get: get a version of an SCCS	get(1)
logname:	get login name	logname(1)
unget: undo a previous	get of an SCCS file	unget(1)
tty:	get the name of the terminal	tty(1)
	getopt: parse command options	getopt(1)
options getopts,	getoptcvt: parse command	getoptcvt(1)
options getopts,	getoptcvt: parse command	getopts(1)
command options	getopts, getoptcvt: parse	getoptcvt(1)
command options	getopts, getoptcvt: parse	getopts(1)
graph: draw a	graph	graph(1G)
	graph: draw a graph	graph(1G)
graphics: access	graphic and numeric commands	graphics(1G)
gutil:	graphic utilities	gutil(1G)
/network useful with	graphical commands	stat(1G)
/erase, hardcopy, tekset, td:	graphical device routines and/	erase(1G)
/erase, hardcopy, tekset, td:	graphical device routines and/	gdev(1G)
/erase, hardcopy, tekset, td:	graphical device routines and/	hardcopy(1G)
/erase, hardcopy, tekset, td:	graphical device routines and/	hpd(1G)
/erase, hardcopy, tekset, td:	graphical device routines and/	td(1G)
/erase, hardcopy, tekset, td:	graphical device routines and/	tekset(1G)
dspst: display process status	graphically	dspst(1)
numeric commands	graphics: access graphic and	graphics(1G)
tplot:	graphics filters	tplot(1G)
	greek: select terminal filter	greek(1)
pattern	grep: search a file for a	grep(1)
chown, chgrp: change owner or	group	chgrp(1)
chown, chgrp: change owner or	group	chown(1)
newgrp: log in to a new	group	newgrp(1)
id: print user and	group IDs and names	id(1)
update, and regenerate	groups of programs/maintain,	make(1)
vine:	grow vines	vine(6)
worm: play the	growing worm game	worm(6)
btlhangman:	guess the word	btlhangman(6)
mastermind: Mastermind	guessing game	mastermind(6)

simple lexical tasks	lex: generate programs for	lex(1)
generate programs for simple	lexical taskslex:	lex(1)
lbr libraries to SR10 archive	librarieslbr2ar: convert	lbr2ar(1)
llib: list installed	libraries	llib(1)
libraries lbr2ar: convert lbr	libraries to SR10 archive	lbr2ar(1)
inlib: install a user-supplied	library	inlib(1)
relation for an object	librarylorder: find ordering	lorder(1)
portable/ ar: archive and	library maintainer for	ar(1)
line: read one	line	line(1)
col: filter reverse	line feeds	col(1)
strip: strip symbol and	line number information from a/	strip(1)
nl:	line numbering filter	nl(1)
out selected fields of each	line of a filecut: cut	cut(1)
send/cancel requests to an LP	line printerlp, cancel:	cancel(1)
send/cancel requests to an LP	line printerlp, cancel:	lp(1)
	line: read one line	line(1)
files comm - select or reject	lines common to two sorted	comm(1)
uniq: report repeated	lines in a file	uniq(1)
of several files or subsequent	lines of one file/same lines	paste(1)
subsequent/ paste: merge same	lines of several files or	paste(1)
ln: create a hard or soft	link	ln(1)
files ld:	link editor for common object	ld(1)
	lint: a C program checker	lint(1)
chacl: change access control	list	chacl(1)
cpacl: copy access control	list	cpacl(1)
lsacl: list access control	list	lsacl(1)
lsacl:	list access control list	lsacl(1)
configure STREAMS/ trconf:	list active Streams or	trconf(1)
ls:	list contents of directory	ls(1)
based access control	list editor/Domain/Dialog	dbacl(1)
llib:	list installed libraries	llib(1)
lty:	list installed types	lty(1)
llkob:	list locked objects	llkob(1)
address space las:	list objects mapped into the	las(1)
nm: print name	list of common object file	nm(1)
from a common object file	list: produce C source listing	list(1)
file list: produce C source	listing from a common object	list(1)
xargs: construct argument	list(s) and execute command	xargs(1)
	llib: list installed libraries	llib(1)
	llkob: list locked objects	llkob(1)
	ln: create a hard or soft link	ln(1)
lcm:	load a color map	lcm(1)
llkob: list	locked objects	llkob(1)
make entries in the system	loglogger:	logger(1)
newgrp:	log in to a new group	newgrp(1)
rwho: who's	logged in on local machines	rwho(1C)
system log	logger: make entries in the	logger(1)
rlogin: remote	login	rlogin(1C)
logname: get	login name	logname(1)
start_sh: start a	log-in shell	start_sh(1)
	login: sign on	login(1)
	logname: get login name	logname(1)
finger: user information	lookup program	finger(1)
for an object library	lorder: find ordering relation	lorder(1)
nice: run a command at	low priority	nice(1)
/pathnames between upper and	lowercase and preserve colons	cvtname(1)

requests to an LP line/	lp, cancel: send/cancel	cancel(1)
requests to an LP line/	lp, cancel: send/cancel	lp(1)
send/cancel requests to an	LP line printerlp, cancel:	cancel(1)
send/cancel requests to an	LP line printerlp, cancel:	lp(1)
disable: enable/disable	LP printersenable,	disable(1)
disable: enable/disable	LP printersenable,	enable(1)
lpstat: print	LP status information	lpstat(1)
information	lpstat: print LP status	lpstat(1)
	ls: list contents of directory	ls(1)
list	lsacl: list access control	lsacl(1)
	lty: list installed types	lty(1)
	m4: macro processor	m4(1)
show host status of local	machinesruptime:	ruptime(1C)
rwho: who's logged in on local	machines	rwho(1C)
m4:	macro processor	m4(1)
rbak: restore or index a	magnetic media backup file	rbak(1)
wbak: create a	magnetic media backup file	wbak(1)
program mt:	magnetic tape manipulating	mt(1)
edmtdesc: edit	magtape descriptor file	edmtdesc(1)
rwmt: read/write foreign	magtapes	rwmt(1)
send mail to users or read	mailmail, rmail:	mail(1)
send mail to users or read	mailmail, rmail:	rmail(1)
file for transmission via	mail/encode/decode a binary	uudecode(1C)
file for transmission via	mail/encode/decode a binary	uuencode(1C)
users or read mail	mail, rmail: send mail to	mail(1)
users or read mail	mail, rmail: send mail to	rmail(1)
mail, rmail: send	mail to users or read mail	mail(1)
mail, rmail: send	mail to users or read mail	rmail(1)
processing system	mailx: interactive message	mailx(1)
regenerate groups of/ make:	maintain, update, and	make(1)
ar: archive and library	maintainer for portable/	ar(1)
SCCS file delta:	make a delta (change) to an	delta(1)
mkapr:	make a problem report	mkapr(1)
mkapr:	make an Apollo product report	mkapr(1)
mkdir:	make directories	mkdir(1)
logger:	make entries in the system log	logger(1)
colors bgcolor:	make interesting background	bgcolor(6)
regenerate groups of programs	make: maintain, update, and	make(1)
banner:	make posters	banner(1)
manual	man: print entries in this	man(1)
intm install a type	manager	intm(1)
dm: Display	Manager Commands	dm(1)
comment section mcs:	manipulate the object file	mcs(1)
mt: magnetic tape	manipulating program	mt(1)
man: print entries in this	manual	man(1)
lcm: load a color	map	lcm(1)
las: list objects	mapped into the address space	las(1)
umask: set file-creation mode	mask	umask(1)
mastermind:	Mastermind guessing game	mastermind(6)
guessing game	mastermind: Mastermind	mastermind(6)
maze: generate a	maze	maze(6)
	maze: generate a maze	maze(6)
file comment section	mcs: manipulate the object	mcs(1)
bcd: convert to antique	media	bcd(6)
restore or index a magnetic	media backup filerbak:	rbak(1)
wbak: create a magnetic	media backup file	wbak(1)

nl: line	numbering filter	nl(1)
number: convert Arabic	numerals to English	number(6)
graphics: access graphic and	numeric commands	graphics(1G)
format/ obj2coff: convert	OBJ format modules to COFF	obj2coff(1)
modules to COFF format/	obj2coff: convert OBJ format	obj2coff(1)
dump selected parts of an	object filedump:	dump(1)
C source listing from a common	object filelist: produce	list(1)
nm: print name list of common	object file	nm(1)
information from a common	object file/and line number	strip(1)
mcs: manipulate the	object file comment section	mcs(1)
ld: link editor for common	object files	ld(1)
sizes in bytes of common	object files/print section	size(1)
find ordering relation for an	object librarylorder:	lorder(1)
crtyobj: create a type	object module for binding	crtyobj(1)
llkob: list locked	objects	llkob(1)
address space las: list	objects mapped into the	las(1)
number factor:	obtain the prime factors of a	factor(1)
od:	octal dump	od(1)
	od: octal dump	od(1)
bldt: display time	operating system was built	bldt(1)
join: relational database	operator	join(1)
getopt: parse command	options	getopt(1)
getoptcvt: parse command	optionsgetopts,	getoptcvt(1)
getoptcvt: parse command	optionsgetopts,	getopts(1)
stty: set the	options for a terminal	stty(1)
object library lorder: find	ordering relation for an	lorder(1)
chown, chgrp: change	owner or group	chgrp(1)
chown, chgrp: change	owner or group	chown(1)
and expand files	pack, pcat, unpack: compress	pack(1)
and expand files	pack, pcat, unpack: compress	pcat(1)
and expand files	pack, pcat, unpack: compress	unpack(1)
crpad: create a transcript	pad and window	crpad(1)
getopt:	parse command options	getopt(1)
getopts, getoptcvt:	parse command options	getoptcvt(1)
getopts, getoptcvt:	parse command options	getopts(1)
information chfn, chsh,	passwd: change password file	chfn(1)
information chfn, chsh,	passwd: change password file	chsh(1)
information chfn, chsh,	passwd: change password file	passwd(1)
chfn, chsh, passwd: change	password file information	chfn(1)
chfn, chsh, passwd: change	password file information	chsh(1)
chfn, chsh, passwd: change	password file information	passwd(1)
several files or subsequent/	paste: merge same lines of	paste(1)
dirname: deliver portions of	path namesbasename,	basename(1)
dirname: deliver portions of	path namesbasename,	dirname(1)
lowercase/ cvtname: convert	pathnames between upper and	cvtname(1)
grep: search a file for a	pattern	grep(1)
processing language awk:	pattern scanning and	awk(1)
egrep: search a file for a	pattern using full regular/	egrep(1)
expand files pack,	pcat, unpack: compress and	pack(1)
expand files pack,	pcat, unpack: compress and	pcat(1)
expand files pack,	pcat, unpack: compress and	unpack(1)
mesg:	permit or deny messages	mesg(1)
ptx:	permuted index	ptx(1)
pg: file	perusal filter for CRTs	pg(1)
CRTs	pg: file perusal filter for	pg(1)
split: split a file into	pieces	split(1)

tee:	pipe fitting	tee(1)
fish:	play Go Fish	fish(6)
mille:	play Mille Bournes	mille(6)
boggle:	play the game of boggle	boggle(6)
worm:	play the growing worm game	worm(6)
and library maintainer for	portable archivesar: archive	ar(1)
basename, dirname: deliver	portions of path names	basename(1)
basename, dirname: deliver	portions of path names	dirname(1)
banner: make	posters	banner(1)
	pr: print files	pr(1)
cpp: the C language	preprocessor	cpp(1)
upper and lowercase and	preserve colons/between	cvtname(1)
cvt_font: convert fonts from	pre-SR10 to SR10 format	cvt_font(1)
unget: undo a	previous get of an SCCS file	unget(1)
by Domain/OS Aegis print/	prf: queue a file for printing	prf(1)
factor: obtain the	prime factors of a number	factor(1)
primes: print	prime numbers	primes(6)
	primes: print prime numbers	primes(6)
btlfortune:	print a random comment	btlfortune(6
interesting, adage fortune:	print a random, hopefully	fortune(6)
prs:	print an SCCS file	prs(1)
date:	print and set the date	date(1)
cal:	print calendar	cal(1)
of a file sum:	print checksum and block count	sum(1)
editing activity sact:	print current SCCS file	sact(1)
man:	print entries in this manual	man(1)
fst:	print fault status information	fst(1)
cat f1 concatenate and	print files	cat(1)
pr:	print files	pr(1)
host system hostid: set or	print identifier of current	hostid(1)
banner:	print large banner on printer	banner(6)
lpstat:	print LP status information	lpstat(1)
trace modules on/ trmon:	print messages collected by	trmon(1)
object file nm:	print name list of common	nm(1)
system hostname: set or	print name of current host	hostname(1
system uname:	print name of current UNIX	uname(1)
news:	print news items	news(1)
primes:	print prime numbers	primes(6)
tb:	print process traceback	tb(1)
of common object files size:	print section sizes in bytes	size(1)
printing by Domain/OS Aegis	print spooler/a file for	prf(1)
names id:	print user and group IDs and	id(1)
banner: print large banner on	printer	banner(6)
requests to an LP line	printer/cancel: send/cancel	cancel(1)
requests to an LP line	printer/cancel: send/cancel	lp(1)
disable: enable/disable LP	printersenable,	disable(1)
disable: enable/disable LP	printersenable,	enable(1)
print/ prf: queue a file for	printing by Domain/OS Aegis	prf(1)
information strinfo:	prints STREAMS-related	strinfo(1)
nice: run a command at low	priority	nice(1)
kill: terminate a	process	kill(1)
wait: await completion of	process	wait(1)
timex: time a command; report	process data and system/	timex(1)
crp: create a	process on a remote node	crp(1)
ps: report	process status	ps(1)
dspst: display	process status graphically	dspst(1)

tb: print	process traceback	tb(1)
awk: pattern scanning and	processing language	awk(1)
mailx: interactive message	processing system	mailx(1)
m4: macro	processor	m4(1)
a common object file list:	produce C source listing from	list(1)
mkapr: make an Apollo	product report	mkapr(1)
	prof: display profile data	prof(1)
prof: display	profile data	prof(1)
ksh: the Korn shell command	programming language	ksh(1)
/Bourne Shell (command	programming language)	rsh(1)
/Bourne Shell (command	programming language)	sh(1)
user interface to the TELNET	protocoltelnet:	telnet(1C)
tftp: trivial file transfer	protocol	tftp(1C)
true, false:	provide truth values	false(1)
true, false:	provide truth values	true(1)
	prs: print an SCCS file	prs(1)
	ps: report process status	ps(1)
	ptx: permuted index	ptx(1)
your screen into a scramble	puzzlescramble: turn	scramble(6)
puzzle:	puzzle game	puzzle(6)
	puzzle: puzzle game	puzzle(6)
	pwd: working directory name	pwd(1)
tput: initialize a terminal or	query terminfo database	tput(1)
Domain/OS Aegis print/ prf:	queue a file for printing by	prf(1)
ipcrm: remove a message	queue, semaphore set, or/	ipcrm(1)
command immune to hangups and	quitsnohup: run a	nohup(1)
	quiz: test your knowledge	quiz(6)
display	rain: animated raindrops	rain(6)
rain: animated	raindrops display	rain(6)
strfile, unstr: create a	random access file for storing/	strfile(6)
btlfortune: print a	random comment	btlfortune(6)
adage fortune: print a	random, hopefully interesting,	fortune(6)
random:	random number generator	random(6)
generator	random: random number	random(6)
fsplit: split FORTRAN or	ratfor files	fsplit(1)
dialect	ratfor: rational FORTRAN	ratfor(1)
ratfor:	rational FORTRAN dialect	ratfor(1)
magnetic media backup file	rbak: restore or index a	rbak(1)
	rcp: remote file copy	rcp(1C)
rmail: send mail to users or	read mailmail,	mail(1)
rmail: send mail to users or	read mailmail,	rmail(1)
line:	read one line	line(1)
rwmt:	read/write foreign magtapes	rwmt(1)
swapul:	rearrange underlining	swapul(1)
ed,	red: text editor	ed(1)
ed,	red: text editor	red(1)
to which the root directory	refers/change the node	rootnode(1)
compile	regcmp: regular expression	regcmp(1)
make: maintain, update, and	regenerate groups of programs	make(1)
SR10 formats cvtrgy: convert	registry between SR9.x and	cvtrgy(1)
regcmp:	regular expression compile	regcmp(1)
file for a pattern using full	regular expressions/search a	egrep(1)
sorted files comm - select or	reject lines common to two	comm(1)
lorder: find ordering	relation for an object library	lorder(1)
join:	relational database operator	join(1)
calendar:	reminder service	calendar(1)

rcp:	remote file copy	rcp(1C)
rlogin:	remote login	rlogin(1C)
crp: create a process on a	remote node	crp(1)
rS:	remote shell	remsh(1C)
file rmdel:	remove a delta from an SCCS	rmdel(1)
semaphore set, or/ ipcrm:	remove a message queue,	ipcrm(1)
rm, rmdir:	remove files or directories	rm(1)
rm, rmdir:	remove files or directories	rmdir(1)
uniq: report	repeated lines in a file	uniq(1)
mkapr: make a problem	report	mkapr(1)
mkapr: make an Apollo product	report	mkapr(1)
communication/ ipcs:	report inter-process	ipcs(1)
timex: time a command;	report process data and system/	timex(1)
ps:	report process status	ps(1)
file uniq:	report repeated lines in a	uniq(1)
lp, cancel: send/cancel	requests to an LP line printer	cancel(1)
lp, cancel: send/cancel	requests to an LP line printer	lp(1)
media backup file rbak:	restore or index a magnetic	rbak(1)
col: filter	reverse line feeds	col(1)
revscr:	reverse screen	revscr(6)
	revscr: reverse screen	revscr(6)
	rlogin: remote login	rlogin(1C)
directories	rm, rmdir: remove files or	rm(1)
directories	rm, rmdir: remove files or	rmdir(1)
read mail mail,	rmail: send mail to users or	mail(1)
read mail mail,	rmail: send mail to users or	rmail(1)
SCCS file	rmdel: remove a delta from an	rmdel(1)
directories rm,	rmdir: remove files or	rm(1)
directories rm,	rmdir: remove files or	rmdir(1)
robots: fight off villainous	robots	robots(6)
robots	robots: fight off villainous	robots(6)
/change the node to which the	root directory refers	rootnode(1)
which the root directory/	rootnode: change the node to	rootnode(1)
/tekset, td: graphical device	routines and filters	erase(1G)
/tekset, td: graphical device	routines and filters	gdev(1G)
/tekset, td: graphical device	routines and filters	hardcopy(1C
/tekset, td: graphical device	routines and filters	hpd(1G)
/tekset, td: graphical device	routines and filters	td(1G)
/tekset, td: graphical device	routines and filters	tekset(1G)
	rS: remote shell	remsh(1C)
Bourne Shell (command/ sh,	rsh: the standard/restricted	rsh(1)
Bourne Shell (command/ sh,	rsh: the standard/restricted	sh(1)
nice:	run a command at low priority	nice(1)
hangups and quits nohup:	run a command immune to	nohup(1)
local machines	ruptime: show host status of	ruptime(1C)
machines	rwho: who's logged in on local	rwho(1C)
magtapes	rwmt: read/write foreign	rwmt(1)
editing activity	sact: print current SCCS file	sact(1)
and iron men	sail: multi-user wooden ships	sail(6)
bfs: big file	scanner	bfs(1)
language awk: pattern	scanning and processing	awk(1)
the delta commentary of an	SCCS deltacdc: change	cdc(1)
comb: combine	SCCS deltas	comb(1)
make a delta (change) to an	SCCS filedelta:	delta(1)
get: get a version of an	SCCS file	get(1)
prs: print an	SCCS file	prs(1)

rmdel: remove a delta from an	SCCS file	rmdel(1)
compare two versions of an	SCCS filesccsdiff:	sccsdiff(1)
undo a previous get of an	SCCS fileunget:	unget(1)
val: validate	SCCS file	val(1)
sact: print current	SCCS file editing activity	sact(1)
admin: create and administer	SCCS files	admin(1)
what: identify	SCCS files	what(1)
subsystem	sccs: front end for the SCCS	sccs(1)
help: ask for	SCCS help	help(1)
sccs: front end for the	SCCS subsystem	sccs(1)
of an SCCS file	sccsdiff: compare two versions	sccsdiff(1)
turn your screen into a	scramble puzzlescramble:	scramble(6)
into a scramble puzzle	scramble: turn your screen	scramble(6)
	scrattr: screen attributes	scrattr(1)
melt: melt the	screen	melt(6)
revscr: reverse	screen	revscr(6)
scrattr:	screen attributes	scrattr(1)
scramble: turn your	screen into a scramble puzzle	scramble(6)
scrto: set/show	screen timeout	scrto(1)
display editor based on/ vi:	screen-oriented (visual)	vi(1)
	scrto: set/show screen timeout	scrto(1)
program	sdiff: side-by-side difference	sdiff(1)
string fgrep:	search a file for a character	fgrep(1)
grep:	search a file for a pattern	grep(1)
using full regular/ egrep:	search a file for a pattern	egrep(1)
the object file comment	sectionmcs: manipulate	mcs(1)
common object/ size: print	section sizes in bytes of	size(1)
	sed: stream editor	sed(1)
to two sorted files comm -	select or reject lines common	comm(1)
greek:	select terminal filter	greek(1)
of a file cut: cut out	selected fields of each line	cut(1)
file dump: dump	selected parts of an object	dump(1)
ipcrm: remove a message queue,	semaphore set, or shared/	ipcrm(1)
mail mail, rmail:	send mail to users or read	mail(1)
mail mail, rmail:	send mail to users or read	rmail(1)
line printer lp, cancel:	send/cancel requests to an LP	cancel(1)
line printer lp, cancel:	send/cancel requests to an LP	lp(1)
/a message queue, semaphore	set, or shared memory id	ipcrm(1)
characteristics kbm:	set/display keyboard	kbm(1)
characteristics tpm:	set/display touchpad and mouse	tpm(1)
settings vsize:	set/display VT100 window	vsize(1)
scrto:	set/show screen timeout	scrto(1)
set/display VT100 window	settingsvsize:	vsize(1)
standard/restricted Bourne/	sh, rsh: the	rsh(1)
standard/restricted Bourne/	sh, rsh: the	sh(1)
queue, semaphore set, or	shared memory id/a message	ipcrm(1)
rS: remote	shell	remsh(1C)
start_sh: start a log-in	shell	start_sh(1)
execute a DM command from the	shellxdmc:	xdmc(1)
language ksh: the Korn	shell command programming	ksh(1)
/the standard/restricted Bourne	Shell (command programming/	rsh(1)
/the standard/restricted Bourne	Shell (command programming/	sh(1)
sail: multi-user wooden	ships and iron men	sail(6)
program sdiff:	side-by-side difference	sdiff(1)
login:	sign on	login(1)
lex: generate programs for	simple lexical tasks	lex(1)

a random access file for	storing strings/unstr: create	strfile(6)
sed:	stream editor	sed(1)
by trace modules on active	Streams/messages collected	trmon(1)
/create initialization code for	STREAMS drivers and modules	mksinit(1)
trace/ trconf: list active	Streams or configure STREAMS	trconf(1)
active Streams or configure	STREAMS trace modules/list	trconf(1)
strinfo: prints	STREAMS-related information	strinfo(1)
random access file for/	strfile, unstr: create a	strfile(6)
STREAMS-related information	strinfo: prints	strinfo(1)
search a file for a character	stringfgrep:	fgrep(1)
random access file for storing	strings/unstr: create a	strfile(6)
number information from a/	strip: strip symbol and line	strip(1)
information from a/ strip:	strip symbol and line number	strip(1)
terminal	stty: set the options for a	stty(1)
another user	su: become super-user or	su(1)
/same lines of several files or	subsequent lines of one file	paste(1)
sccs: front end for the SCCS	subsystem	sccs(1)
count of a file	sum: print checksum and block	sum(1)
du:	summarize disk usage	du(1)
su: become	super-user or another user	su(1)
interval sleep:	suspend execution for an	sleep(1)
	swapul: rearrange underlining	swapul(1)
display address of external	symbolesa:	esa(1)
information from/ strip: strip	symbol and line number	strip(1)
	sync: forces write to disk	sync(1)
	systype: display version stamp	systype(1)
tabs: set	tabs on a terminal	tabs(1)
	tabs: set tabs on a terminal	tabs(1)
a file	tail: deliver the last part of	tail(1)
tar:	tape file archiver	tar(1)
mt: magnetic	tape manipulating program	mt(1)
	tar: tape file archiver	tar(1)
programs for simple lexical	taskslex: generate	lex(1)
	tb: print process traceback	tb(1)
/hpd, erase, hardcopy, tekset,	td: graphical device routines/	erase(1G)
/hpd, erase, hardcopy, tekset,	td: graphical device routines/	gdev(1G)
/hpd, erase, hardcopy, tekset,	td: graphical device routines/	hardcopy(1G)
/hpd, erase, hardcopy, tekset,	td: graphical device routines/	hpd(1G)
/hpd, erase, hardcopy, tekset,	td: graphical device routines/	td(1G)
/hpd, erase, hardcopy, tekset,	td: graphical device routines/	tekset(1G)
teachgammon:	teach the game of backgammon	teachgammon(6)
backgammon	teachgammon: teach the game of	teachgammon(6)
	tee: pipe fitting	tee(1)
gdev: hpd, erase, hardcopy,	tekset, td: graphical device/	erase(1G)
gdev: hpd, erase, hardcopy,	tekset, td: graphical device/	gdev(1G)
gdev: hpd, erase, hardcopy,	tekset, td: graphical device/	hardcopy(1G)
gdev: hpd, erase, hardcopy,	tekset, td: graphical device/	hpd(1G)
gdev: hpd, erase, hardcopy,	tekset, td: graphical device/	td(1G)
gdev: hpd, erase, hardcopy,	tekset, td: graphical device/	tekset(1G)
telnet: user interface to the	TELNET protocol	telnet(1C)
TELNET protocol	telnet: user interface to the	telnet(1C)
functions of the DASI 450	terminal450: handle special	450(1)
emt: emulate a dumb	terminal	emt(1)
stty: set the options for a	terminal	stty(1)
tabs: set tabs on a	terminal	tabs(1)
tty: get the name of the	terminal	tty(1)

animate worms on a display	terminalworms:	worms(6)
flake: induce	terminal dandruff	flake(6)
vt100: VT100	terminal emulator	vt100(1)
greek: select	terminal filter	greek(1)
database tput: initialize a	terminal or query terminfo	tput(1)
functions of DASI 300 and 300s	terminals/handle special	300(1)
functions of Hewlett-Packard	terminalshp: handle special	hp(1)
kill:	terminate a process	kill(1)
initialize a terminal or query	terminfo databasetput:	tput(1)
command	test: condition evaluation	test(1)
quiz:	test your knowledge	quiz(6)
ed, red:	text editor	ed(1)
ex:	text editor	ex(1)
ed, red:	text editor	red(1)
casual users) edit:	text editor (variant of ex for	edit(1)
change the format of a	text filenewform:	newform(1
translate status code value to	text messagestcode:	stcode(1)
protocol	tftp: trivial file transfer	tftp(1C)
ttt:	tic-tac-toe	ttt(6)
execute commands at a later	timeat, batch:	at(1)
execute commands at a later	timeat, batch:	batch(1)
time:	time a command	time(1)
data and system/ timex:	time a command; report process	timex(1)
built bldt: display	time operating system was	bldt(1)
display the module name and	time stampts:	ts(1)
	time: time a command	time(1)
tz: set or display system	time zone	tz(1)
scrto: set/show screen	timeout	scrto(1)
update access and modification	times of a filetouch:	touch(1)
process data and system/	timex: time a command; report	timex(1)
tsort:	topological sort	tsort(1)
modification times of a file	touch: update access and	touch(1)
tpm: set/display	touchpad and mouse/	tpm(1)
	tplot: graphics filters	tplot(1G)
mouse characteristics	tpm: set/display touchpad and	tpm(1)
query terminfo database	tput: initialize a terminal or	tput(1)
	tr: translate characters	tr(1)
Streams or configure STREAMS	trace modules/list active	trconf(1)
/print messages collected by	trace modules on active/	trmon(1)
tb: print process	traceback	tb(1)
crpad: create a	transcript pad and window	crpad(1)
ftp: ARPANET file	transfer program	ftp(1C)
tftp: trivial file	transfer protocol	tftp(1C)
tr:	translate characters	tr(1)
text message stcode:	translate status code value to	stcode(1)
within a font tr_font:	transliterate characters	tr_font(1)
/a binary file for	transmission via mail	uudecode(
/a binary file for	transmission via mail	uuencode(
configure STREAMS trace/	trconf: list active Streams or	trconf(1)
	trek: trekkie game	trek(6)
trek:	trekkie game	trek(6)
characters within a font	tr_font: transliterate	tr_font(1)
tftp:	trivial file transfer protocol	tftp(1C)
collected by trace modules on/	trmon: print messages	trmon(1)
battlestar: a	tropical adventure game	battlestar(
values	true, false: provide truth	false(1)

values	true, false: provide truth	true(1)
true, false: provide	truth values	false(1)
true, false: provide	truth values	true(1)
and time stamp	ts: display the module name	ts(1)
	tsort: topological sort	tsort(1)
	ttt: tic-tac-toe	ttt(6)
terminal	tty: get the name of the	tty(1)
crty: create a new	type	crty(1)
dlty: delete a	type	dlty
file: determine file	type	file(1)
inty: install a new	type	inty(1)
intm install a	type manager	intm(1)
crtyobj: create a	type object module for binding	crtyobj(1)
lty: list installed	types	lty(1)
and slides mmt, mvt:	typeset documents, viewgraphs,	mmt(1)
and slides mmt, mvt:	typeset documents, viewgraphs,	mvt(1)
zone	tz: set or display system time	tz(1)
mask	umask: set file-creation mode	umask(1)
UNIX system	uname: print name of current	uname(1)
swapul: rearrange	underlining	swapul(1)
file unget:	undo a previous get of an SCCS	unget(1)
an SCCS file	unget: undo a previous get of	unget(1)
a file	uniq: report repeated lines in	uniq(1)
	units: conversion program	units(1)
execution uux:	UNIX-to-UNIX system command	uux(1C)
uucp, uulog, uuname:	UNIX-to-UNIX system copy	uucp(1C)
uucp, uulog, uuname:	UNIX-to-UNIX system copy	uulog(1C)
uucp, uulog, uuname:	UNIX-to-UNIX system copy	uuname(1C)
uuto, uupick: public	UNIX-to-UNIX system file copy	uupick(1C)
uuto, uupick: public	UNIX-to-UNIX system file copy	uuto(1C)
files pack, pcat,	unpack: compress and expand	pack(1)
files pack, pcat,	unpack: compress and expand	pcat(1)
files pack, pcat,	unpack: compress and expand	unpack(1)
file for storing/ strfile,	unstr: create a random access	strfile(6)
times of a file touch:	update access and modification	touch(1)
of programs make: maintain,	update, and regenerate groups	make(1)
/convert pathnames between	upper and lowercase and/	cvtname(1)
du: summarize disk	usage	du(1)
stat: statistical network	useful with graphical commands	stat(1G)
become super-user or another	usersu:	su(1)
write: write to another	user	write(1)
id: print	user and group IDs and names	id(1)
crontab:	user crontab file	crontab(1)
program finger:	user information lookup	finger(1)
protocol telnet:	user interface to the TELNET	telnet(1C)
whois: DARPA Internet	username directory service	whois(1)
(variant of ex for casual	users)edit: text editor	edit(1)
wall: write to all	users	wall(1)
mail, rmail: send mail to	users or read mail	mail(1)
mail, rmail: send mail to	users or read mail	rmail(1)
inlib: install a	user-supplied library	inlib(1)
/search a file for a pattern	using full regular expressions	egrep(1)
gutil: graphic	utilities	gutil(1G)
control uustat:	uucp status inquiry and job	uustat(1C)
UNIX-to-UNIX system copy	uucp, uulog, uuname:	uucp(1C)
UNIX-to-UNIX system copy	uucp, uulog, uuname:	uulog(1C)

NAME

intro – introduction to commands

DESCRIPTION

This section describes publicly accessible commands for general utility. In addition, some special commands for communication purposes are described. All commands are listed in alphabetic order, and each is suffixed by ''(1)'' to help identify its place in the *SysV Command Reference.*

N.B.: Commands that relate to system maintenance, distinguished by (1M) in earlier UNIX System documentation, are described in *Managing SysV System Software.*

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of ''normal'' termination) one supplied by the program [see **wait**(2) and **exit**(2)]. The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, or bad or inaccessible data. It is called variously ''exit code'', ''exit status'', or ''return code'', and is described only where special conventions are involved.

BUGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings,and become confused at encountering a null character (the string terminator) within a line.

SEE ALSO

Section 6 (for computer games), *Getting Started with Domain/OS*, and *Using Your SysV Environment.*

NAME

domain – Domain/OS-specific commands and extensions

DESCRIPTION

While providing all of the significant capacity of System V Release 3, Domain/OS SysV actually represents only a subset of the greater capacity of Domain/OS. Furthermore, Domain/OS SysV omits some features of System V that are irrelevant to Apollo® workstations. The following paragraphs describe aspects that are visible to the Domain/OS SysV user and summarize features of System V not implemented under Domain/OS SysV.

Domain/OS Additions to SysV

Pages that describe the Domain/OS-specific commands are identified with the page heading "Domain/OS SysV;" pages describing standard System V UNIX commands are "SysV".

The /usr/apollo/bin Directory

All systems, even if they only have the Aegis environment, now have a **/usr/apollo** directory. It contains certain Domain extensions to the UNIX environment. It also includes C include files for Domain system calls, as well as other added-value files.

The **/usr/apollo/bin** directory contains Domain commands that apply to all three environments or extend the UNIX environment. The following Domain/OS-specific commands appear in **/usr/apollo/bin**:

bldt	Displays information about the version of Domain/OS.
chacl	Changes the entries in an object's access control list (ACL).
cpacl	Copies access control lists (ACLs).
cpscr	Copies the current screen image, without clearing it, to the file you specify.
crddf	Creates, displays, or modifies a device descriptor file (DDF).
crp	Creates a process on a remote node.
crpad	Creates a transcript pad, copies a file (or standard input) into that pad, and then opens a window into the pad.
crty	Creates a new type. It creates an identifier for the new type, and associates it with the supplied type name. New types are used to identify a new kind of manager for streams.
crtyobj	Creates an object module that contains a global symbol with the type UID. This module is bound with type managers. The variable is passed into calls to **trait_$mgr_dcl** to declare support for the specified type.
cvt_font	Converts SR9 font files to the new font format for SR10.

cvtname	Converts SR9 pathnames between upper and lowercase and preserve colons.
cvtrgy	Allows the system administrator to generate an SR10 format registry database from SR9.7 registry files, or generates SR9.7 registry files with data from the SR10 registry.
dbacl	Provides an interactive menu-based editor for manipulating Access Control Lists (ACLs).
dde	Allows you to load and debug programs written in any programming language supported by the Domain/OS operating system, including assembler.
dlty	Deletes a type and any installed type manager.
dm	Contains a list of Display Manager commands.
dspst	Displays process statistics in a graphical, bar-chart fashion within the current process window.
edfont	Allows you to create, edit and view character font files.
edmtdesc	Allows you to create, list, and modify the magnetic tape descriptor object.
emt	Allows your node to emulate an ASCII terminal connected to another computer. This asynchronous connection exists through a stream opened on one of the node's SIO lines. **emt** also permits ASCII file transfer between your node and the remote host.
esa	Displays the address of an external symbol in an installed library. This command is primarily intended for system-level debugging.
fst	Prints information about the most recent fault that occurred in the process.
hpc	The hpc (histogram_program_counter), part of Domain/PAK (Domain Performance Analysis Kit), looks at the performance of programs at the PC level.
intm	Installs a type manager for the *type_name*.
inty	Installs a type from one node to another.
iso	Converts files written with the overloaded 7-bit national fonts to the International Standards Organization (ISO) 8-bit format. This includes: **french_to_iso**, **german_to_iso**, **nor.dan_to_iso**, **swedish_to_iso**, **swiss_to_iso**, and **uk_to_iso**
kbm	Allows you to set the characteristics for the keyboard.
las	Produces a list of objects mapped into the address space.

lbr2ar	Converts pre-SR10 **lbr** library files containing object modules in OBJ format to SR10 ar library archive files containing object modules in COFF format.
lcm	Loads a color map from a file that specifies a set of color map entries.
llib	Lists those libraries which have been installed in the current process via the build-in inlib shell.
llkob	Lsts the locked objects resident on volumes mounted on this node, and objects resident in other nodes that are locked by processes running locally.
lsacl	Shows the access control list (ACL) associated with the files and directories specified.
lty	Lists the types currently installed on a volume.
mkapr	Makes an Apollo product report.
obj2coff	Converts SR9.5 or later object format modules to SR10 COFF format modules. Either individual modules, or complete bound programs may be converted.
prf	Queues a file for printing.
rbak	Restores objects from the backup input media written by **wbak** (write_backup). The backup input media can be magnetic media, file or standard input.
rwmt	Reads tapes from non-Domain installations and writes tapes that can be read by non-Domain installations. **rwmt** can read and write u nlabeled tapes, as well as ANSI level 1-4 labeled tapes.
scrattr	Lists the *X* and *Y* dimensions of the display in pixels.
scrto	Sets or displays the number of minutes the system waits before it shuts off the display screen. It begins counting minutes after the last input event or window configuration change.
stcode	Prints the text message associated with a hexadecimal status code.
tb	Prints a process traceback, listing the name and current line number of each routine on the call stack.
tpm	Allows you to define characteristics for the touchpad and mouse.
tr_font	Allows you to change the order in which characters appear in fonts.
ts	Displays the time stamp and module name stored in an object module.
tz	Sets the system time zone to a known time zone or to an offset from Coordinate Universal Time (utc).

vsize	Allows you to set the dimensions of the VT100 emulator window pane. This command is valid only from within the VT100 emulator (which is invoked with the VT100 command); attempting to use it directly from the shell causes an error.
vt100	Creates a window running the VT100 terminal emulator and starts up a shell within the window.
wbak	Writes one or more objects to either a removable media, disk file or standard output.
xdmc	Allows you to invoke Display Manager commands from the command shell or from within a shell script.

Domain/OS SysV Extensions

This section describes Apollo extensions to standard UNIX commands.

ar	Domain/OS SysV **ar** builds a module name table and a long name table in addition to the symbol table; these tables are stored in files that are never mentioned or accessible. This makes **ranlib** obsolete.
cc	cc is the Domain/OS SysV interface to the preprocessor (**cpp**), the Domain C compiler, and the link editor (**ld**). The Domain/OS SysV cc command provides some unique options; not all standard UNIX options are available. The −A option identifies a unique set of Domain/OS extensions to cc and **ld**.
cp	The **cp** command includes a number of Domain/OS extensions. See **cp**(1) for a complete description of these extensions.
csh	**limit −h resource maximum-use**. You cannot use limit to set the stack size, and the coredumpsize limit is always 0 in Domain/OS. **path**. The default search path in Domain/OS SysV is (. **/usr/ucb /bin /usr/bin /usr/apollo/bin**). However, this may vary from system to system. For the super-user, the default search path is (**/etc /bin /usr/bin /usr/apollo/bin**), which may also vary.
ksh	Domain/OS SysV includes support for the Korn shell and adds some extensions to this shell. See "UNIX Shell Extensions", below for a brief summary of the added features.
ld	The Domain/OS SysV version of **ld** includes support for features that are not available on System V Release 3. Domain/OS **ld** supports the following extensions: static resource information records (**.sri**), module information records (**.mir**), and control of global variable visibility. The −A option identifies a unique set of Domain/OS SysV extensions to **cc** and **ld**.
ln	Symbolic links in Domain/OS are implemented as soft links. These are identical in behavior, except that soft links to not have protections associated with the links themselves.

login	The **login** command is a merge of the System V and 4.3BSD **login** commands. The **−p** argument causes the remainder of the environment to be preserved, otherwise any previous environment is discarded. Domain/OS **login** includes new security features for dial-up lines, **/etc/d_users** and **/etc/d_passwd**. **/etc/d_users** is simply a file containing a list of users authorized to log in on this node. **/etc/d_passwd** is a file containing lines which specify a user's log-in shell, and the dial-in password for the specified shell as returned by **crypt**(3). If an entry for the user's log-in shell is not found in this file, the password for **/bin/sh** is used.
lorder	The need for **lorder** has vanished on Domain/OS systems, since **ar**(1) and **ld**(1) cooperate to create randomly accessed libraries.
ls	If you specify **−T** with the **−l** option, it shows the Domain/OS "type" of each file.
mkdir	The mechanism for assigning the initial file ACL and initial directory ACL for the directory created by the **mkdir** command has been changed. The initial file ACL and initial directory ACL are now inherited from the parent directory.
nm	The **−Ag** option checks KGT (Known Global Table) to see if undefined globals are defined in global libraries. If specified with the **−u** option, **nm** will not print those undefined symbols that are defined in global libraries.
passwd	On Domain/OS systems, the **/etc/passwd** file is a typed file, which is automatically generated by the registry daemon. The registry administrator can make the person information in the registry read-only, in which case normal users cannot change the "Name" field.
ps	The Domain/OS *nodename* option shows information about processes running on the specified node.
ranlib	**ranlib** is not necessary on Domain/OS SysV systems; however, it is provided as a no-op for compatibility.
strip	The **−Aa** option strips all debugging information, including that needed for traceback. The **.blocks** and **.lines** sections, if present, will be removed. This option strips more information than the default strip behavior, and is added for users who wish to remove all symbolic information.
tftp	The Domain/OS SysV versions of **tftp** and **tftpd**(1M) are adaptations of the MIT Project Athena implementations of the **tftp** protocol. Domain/OS SysV **tftp** will interface with any RFC783 compliant implementation.

uucp	Domain/OS SysV supports "HoneyDanBer" uucp for both the SysV and SysV environments. (See "UUCP Support" below).
who	The **who** command includes a number of Domain/OS extensions.

UNIX Shell Extensions

Domain/OS includes support for the additional shell built-in commmands **inlib**, **rootnode**, and **ver**.

rootnode causes / to refer to the node entry directory of //**nodename** instead of the current node entry directory.

inlib installs a library at the current shell level; it remains installed until the shell that installed it exits. The newly installed library will be used to resolve external references of programs (and libraries) loaded after its installation.

ver changes, temporarily or permanently, the UNIX version of commands that are executed by the shell. The command also displays the version in use.

csh and **sh** also include support for a new command line option, –D*name*=*value*. This option sets the parameter *name* to *value*, then passes it to the shell's environment. This option is useful for tailoring the environment of a shell invoked from a program that isn't a shell (such as the Display Manager).

ksh has also been extended to support editing commands in Display Manager pads. If the value of the variable **EDITOR** ends in **emacs**, **gmacs**, or **vi** and the **VISUAL** is not set, the corresponding option is turned on. This value should be unset for shells running in Apollo transcript pads.

For Apollo transcript pads, the variable **FCEDIT** should be set to 'pad'. With dialup lines or in VT100 windows, values like **vi** or **emacs** are useful.

The in-line editing options are not useful in Apollo transcript pads. The command input pane associated with Transcript pads allows full command line editing. Setting **VISUAL** or **EDITOR** in Apollo transcript pads causes the pad to flip in and out of raw mode.

TCP/IP Support

At SR10, Domain/OS supports TCP/IP in the Aegis, BSD and SysV environments. One version operates in all environments.

Unsupported Commands

The following commands from System V are not supported.

4014	**bs**	**ct**
cw	**eqn**	**ged**
hpio	**ismpx**	**jterm**
jwin	**login**	**mm**
nroff	**sag**	**sar**
shl	**sysadm**	**tbl**
toc	**troff**	

At this revision **nroff**, **troff**, **tbl**, and associated text processing tools are bundled and sold as a separate package for System V users.

SEE ALSO

intro(1), domain(1M), domain(7),
Using Your SysV Environment;
Managing SysV System Software.

NAME

300, 300s – handle special functions of DASI 300 and 300s terminals

SYNOPSIS

300 [**+12**] [**–n**] [**–dt,l,c**]

300s [**+12**] [**–n**] [**–dt,l,c**]

DESCRIPTION

The **300** command supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal; **300s** performs the same functions for the DASI 300s (GSI 300s or DTC 300s) terminal. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. In the following discussion of the **300** command, it should be noted that unless your system contains the DOCUMENTER'S WORKBENCH Software, references to certain commands (e.g., **nroff, neqn, eqn,**) will not work. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12-pitch text. It reduces printing time 5 to 70%. You can use the **300** command to print equations neatly, in the sequence:

neqn file . . . | nroff | 300

OPTIONS

+12 Permits use of 12-pitch, 6 lines/inch text. DASI 300 terminals normally allow only two combinations: 10-pitch, 6 lines/inch, or 12-pitch, 8 lines/inch. To obtain the 12-pitch, 6 lines per inch combination, turn the PITCH switch to 12, and use the **+12** option.

–n Controls the size of half-line spacing. A half-line is, by default, equal to 4 vertical plot increments. Because each increment equals 1/48 of an inch, a 10-pitch line-feed requires 8 increments, while a 12-pitch line-feed needs only 6. The first digit of **n** overrides the default value, thus allowing for individual taste in the appearance of subscripts and superscripts. For example, **nroff** half-lines can be made to act as quarter-lines by using –2. You can also obtain appropriate half-lines for 12-pitch, 8 lines/inch mode by using the option –3 alone, having set the PITCH switch to 12-pitch.

–d*t*,*l*,*c* controls delay factors. The default setting is **–d3,90,30**. DASI 300 terminals sometimes produce peculiar output when faced with very long lines, too many tab characters, or long strings of blankless, non-identical characters. One null (delay) character is inserted in a line for every set of *t* tabs, and for every contiguous string of *c* non-blank, non-tab characters. If a line is longer than *l* bytes, 1+(total length)/20 nulls are inserted at the end of that line. Items can be omitted from the end of the list, implying use of the default values. Also, a value of zero for *t* (*c*) results in two null bytes per tab (character). The former may be needed for C programs, the latter for files like **/etc/passwd**. Because terminal behavior varies according to the specific characters printed and the load on a system, you may have to experiment with these values to get correct output. The **−d** option exists only as a last resort for those few cases that do not otherwise print properly. For example, you can print the file /etc/passwd using **−d3,30,5**. The value **−d0,1** is a good one to use for C programs that have many levels of indentation.

You can use **300** with the **nroff −s** flag or **.rd** requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of pressing RETURN in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the following sequences are equivalent:

nroff −T300 files ... and **nroff files .. | 300**
nroff −T300−12 files ... and **nroff files .. | 300 +12**

Using **300** can often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of **300** can produce better-aligned output.

The **neqn** names of, and resulting output for, the Greek and special characters supported by **300** are shown in **greek**(5).

NOTE

The delay control interacts heavily with the prevailing carriage return and line-feed delays. The **stty**(1) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

WARNING

If your terminal has a PLOT switch, make sure it is turned ON before **300** is used.

BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.
If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

SEE ALSO

450(1), mesg(1), graph(1G), stty(1), tabs(1), tplot(1G).
greek(5) in the *SysV Programmer's Reference*.

NAME

450 – handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

The **450** command supports special functions of, and optimizes the use of, the DASI 450 terminal, or any terminal that is functionally identical, such as the Diablo 1620 or Xerox 1700. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols in the same manner as **300**(1). Use **450** to print equations neatly, in the sequence:

```
neqn file ... | nroff | 450
```

Use **450** with the **nroff** –s flag or **.rd** requests when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of RETURN in these cases, you must use the line-feed key to get any response.

Frequently, you can eliminate using **450** in favor of one of the following:

```
nroff –T450 files ...
```

or

```
nroff –T450–12 files ...
```

Using **450** can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of **450** can produce better-aligned output.

neqn names of, and resulting output for, the Greek and special characters supported by **450** are shown in **greek**(5).

NOTE

Unless your system contains DOCUMENTER'S WORKBENCH Software, certain commands (e.g., **eqn**, **nroff**, **tbl**) will not work.

WARNING

Make sure the PLOT switch on your terminal is ON before you use **450**. The SPACING switch should be put in the desired position (either 10– or 12–pitch). In either case, vertical spacing is 6 lines/inch, unless dynamically changed to 8 lines/inch by an appropriate escape sequence.

BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.
If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

SEE ALSO

300(1), mesg(1), stty(1), tabs(1), graph(1G), tplot(1G).
eqn(1), nroff(1), tbl(1) in the *UNIX System V Documentor's Workbench Reference Manual.*
greek(5) in the *SysV Programmer's Reference.*

NAME

admin – create and administer SCCS files

SYNOPSIS

admin [–**n**] [–**i**[*name*]] [–**r***rel*] [–**t**[*name*]] [–**f***flag*[*flag-val*]] [–**d***flag*[*flag-val*]] [–**a***login*] [–**e***login*] [–**m**[*mrlist*]] [–**y**[*comment*]] [–**h**] [–**z**] *files*

DESCRIPTION

admin creates new SCCS files and changes parameters of existing ones. *Options* to **admin** can appear in any order and must be preceded by a dash (–), and named files (note that SCCS file names must begin with the characters s.). If a named file does not exist, it is created, and its parameters are initialized according to the specified *options*. Parameters not initialized by an *option* are assigned a default value. If a named file does exist, parameters corresponding to specified *options* are changed, and other parameters are left as is.

If a directory is named, **admin** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

OPTIONS

–n Indicates that a new SCCS file is to be created.

–**i***[name]* The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see –**r** *option* for delta numbering scheme). If **i** is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this *option* is omitted, then the SCCS file is created empty. Only one SCCS file can be created by an **admin** command on which the **i** *option* is supplied. Using a single **admin** to create two or more SCCS files requires that they be created empty (no –**i** *option*). Note that the –**i** *option* implies the –**n** *option*.

–**r***rel* The *rel*ease into which the initial delta is inserted. This can only be used if –**i** is also used. If –**r** is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

–**t***[name]* The *name* of a file from which descriptive text for the SCCS file is to be taken. If –**t** is used and **admin** is creating a new SCCS file (the –**n** and/or –**i** *options* also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) using –**t** without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) using –**t** with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

–fflag Specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. You can use several –f's on a single **admin** command line. The allowable *flag*s and their values are:

b Allows use of **–b** on a **get**(1) command to create branch **deltas**.

c*ceil* The highest release (i.e., ''ceiling''), a number greater than 0 but less than or equal to 9999, which can be retrieved by a **get**(1) command for editing. The default value for an unspecified **c** flag is 9999.

f*floor* The lowest release (i.e., ''floor''), a number greater than 0 but less than 9999, which can be retrieved by a **get**(1) command for editing. The default value for an unspecified **f** flag is 1.

d*SID* The default delta number (SIDs+1) to be used by a **get**(1) command.

i*[str]* Causes the "No id keywords (ge6)" message issued by **get**(1) or **delta**(1) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords [see **get**(1)] are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must exactly match the given string, however the string must contain a keyword, and no embedded newlines.

j Allows concurrent **get**(1) commands for editing on the same SIDs+1 of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

l*list* A *list* of releases to which deltas can no longer be made (get –e against one of these ''locked'' releases fails). The *list* has the following syntax:

<list> ::= *<range>* | *<list>* , *<range>*
<range>˜::= | **a**

The character **a** in the *list* is equivalent to specifying *all releases* for the named SCCS file.

n Causes **delta**(1) to create a ''null'' delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as ''anchor points'' so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

q*text* User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by ge (1).

m*mod* *Mod*ule name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by get(1). If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading s. removed.

t*type* *Type* of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by get(1).

v*pgm* Causes **delta**(1) to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validity checking program [see **delta**(1)]. (If this flag is set when creating an SCCS file, the **m** *option* must also be used even if its value is null).

–d*flag* Causes removal (deletion) of the specified *flag* from an SCCS file. The **–d** *option* can be specified only when processing existing SCCS files. Several **–d** *options* can be supplied on a single **admin** command. See the –f *option* for allowable *flag* names.

l*list* A *list* of releases to be "unlocked". See the –f *option* for a description of the **l** flag and the syntax of a *list*.

–a*login* A *login* name, or numerical UNIX system group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several **a** *options* can be used on a single **admin** command line. As many *login*s, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, anyone can add deltas. If *login* or group ID is preceded by a ! they are to be denied permission to make deltas.

–e*login* A *login* name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. You can use several –e's on a single **admin** command line.

–m*[mrlist]* The list of Modification Requests (MR) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to **delta**(1). The v flag must be set and the MR numbers are validated if the v flag has a value (the name of an MR number validation program). Diagnostics occur if the v flag is not set or MR validation fails.

−y*[comment]*	The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of **delta**(1). Omission of the −y *option* results in a default comment line being inserted in the form: date and time created *YY/MM/DD HH:MM:SS* by *login* The −y *option* is valid only if the **−i** and/or **−n** *options* are specified (i.e., a new SCCS file is being created).
−h	Causes **admin** to check the structure of the SCCS file [see **sccsfile**(5)], and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced. This *option* inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.
−z	The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see **−h**, above). Note that use of this *option* on a truly corrupted file can prevent future detection of the corruption.

The last component of all SCCS file names must be of the form **s.file-name**. New SCCS files are given mode 444 [see **chmod**(1)]. Write permission in the pertinent directory is, of course, required to create a file. All writing done by **admin** is to a temporary **x-file**, called **x.file-name**, [see **get**(1)], created with mode 444 if the **admin** command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of **admin**, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of **ed**(1). *Care must be taken!* The edited file should *always* be processed by an **admin −h** to check for corruption followed by an **admin** −z to generate a proper check-sum. Another **admin −h** is recommended to ensure the SCCS file is valid.

admin also makes use of a transient lock file (called **z.file-name**), which is used to prevent simultaneous updates to the SCCS file by different users. See **get**(1) for further information.

FILES

g–file	Existed before the execution of **delta**; removed after completion of **delta**.
p–file	Existed before the execution of **delta**; may exist after completion of **delta**.
q–file	Created during the execution of **delta**; removed after completion of **delta**.
x–file	Created during the execution of **delta**; renamed to SCCS file after completion of **delta**.
z–file	Created during the execution of **delta**; removed during the execution of **delta**.
d–file	Created during the execution of **delta**; removed after completion of **delta**.
/usr/bin/bdiff	Program to compute differences between the ''gotten'' file and the g-file.

DIAGNOSTICS

Use **help**(1) for explanations.

SEE ALSO

delta(1), get(1), prs(1), sccs(1), what(1), sccsfile(4).
ed(1), help(1) in the *Using Your SysV Environment.*

NAME

ar – archive and library maintainer for portable archives

SYNOPSIS

ar *key* [*posname*] *afile* [*name*] . . .

DESCRIPTION

ar maintains groups of files combined into a single archive file. Although its main use is to create and update library files as used by the link editor, **ar** can be used for any similar purpose. The magic string and the file headers used by **ar** consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

When **ar** creates an archive, it produces headers in a format that is portable across all machines. The portable archive format and structure is described in detail in **ar**(4). The link editor uses the archive symbol table to effect multiple passes over libraries of object files in an efficient manner. The link editor is further described in **ld**(1).

ar creates and maintains an archive symbol table and module name table only when there is at least one object file in the archive. The archive symbol table is in a specially named file which is always the first file in the archive. This file is never mentioned or accessible. Whenever **ar** creates or updates the contents of such an archive, it also rebuilds the symbol table. Domain/OS SysV **ar** builds a module name table and a long name table in addition to the symbol table; these tables are stored in files that are never mentioned or accessible.

key is an optional dash (–) followed by one character from the **drqtpmx** set, optionally concatenated with one of more characters from the **vuaibcls** set. *posname* is the name of an optional positioning character. *afile* is the archive file.

OPTION

A This *option* may or may not begin with a dash (–), and is used with the **mxtd** keys to move, extract, list or delete by module name.

KEY CHARACTERS

d Deletes named files from the archive file.

r Replaces named files in the archive file. If the optional character **u** is used with **r**, then only those files with dates of modification later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the **posname** argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) **posname.** Otherwise new files are placed at the end.

q Quickly appends named files to the end of the archive file. Optional positioning characters are invalid. Do not check whether the added members are already in the archive. Useful for avoiding quadratic behavior when creating a large archive piece-by-piece. Unchecked, the file can grow exponentially up to the second degree.

t Prints a table of contents of the archive file. If no names are given, table all files. If names are given, table only those files named.

p Prints named files in the archive.

m Moves named files to the end of the archive. If a positioning character is present, then the **posname** argument must be present and, as in **r**, specifies where the files are to be moved.

x Extracts named files. If no names are given, all files in the archive are extracted. In neither case does x alter the archive file.

KEY ARGUMENTS

v Gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, give a long listing of all information about the files. When used with x, precede each file with a name.

c Creates **afile** and suppress the message produced by default when **afile** is created.

l Places temporary files in the local (current working) directory rather than in the default temporary directory, **TMPDIR**.

s Forces the regeneration of the archive symbol table even if **ar**(1) is not invoked with a command that modifies the archive contents. Useful for restoring the archive symbol table after the **strip**(1) command has been used on the archive.

NOTES

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

FILES

$TMPDIR/*	Temporary files

$TMPDIR is usually /usr/tmp but can be redefined by setting the environment variable **TMPDIR** [see **tempnam()** in **tmpnam(3S)**].

SEE ALSO

ld(1), lorder(1), strip(1), tmpnam(3S), a.out(4), ar(4) in the *SysV Programmer's Reference*.

NAME

asa – interpret ASA carriage control characters

SYNOPSIS

asa [*files*]

DESCRIPTION

Asa interprets the output of FORTRAN programs that use ASA carriage control characters. It processes either the *files* whose names are given as arguments or the standard input if no filenames are supplied. The first character of each line is assumed to be one of the following control characters:

' '	(Blank) Single newline before printing;
0	Double newline before printing;
1	New page before printing;
+	Overprint previous line.

If a line begins with anything other than the above characters, asa automatically interprets it as beginning with a ' ', and produces an appropriate diagnostic on the standard error. It never prints the first character of a line, and it always forces the first line of each input file to start on a new page.

SEE ALSO

ratfor (1).

NAME

at, batch – execute commands at a later time

SYNOPSIS

at *time* [*date*] [+ *increment*]
at –r*job*...
at –l [*job* ...]

batch

DESCRIPTION

at and **batch** read commands from standard input to be executed at a later time. **at** allows you to specify when the commands should be executed, while jobs queued with **batch** execute when system load level permits.

Standard output and standard error output are mailed to you, unless you redirect them elsewhere. Shell environment variables, current directory, umask, and ulimit are retained when you execute either **at** or **batch**. Open file descriptors, traps, and priority are lost.

You can use **at** if your name appears in the file **/usr/lib/cron/at.allow**. If that file does not exist, the file **/usr/lib/cron/at.deny** determines whether or not you are allowed to use **at**. If neither file exists, only **root** can submit a job. The allow/deny files consist of one user name per line. These files can only be modified by the superuser.

batch submits a batch job. It is equivalent to the command **at now** with the exceptions that **batch** goes into a differenct queue and responds earlier with error messages.

OPTIONS

The following options apply to **at** only:

[*time*] [+ *increment*]
Specify *time* when commands are to be executed. One- and two-digit numbers indicate hours, four-digit numbers show hours and minutes. You may alternately specify the time as two numbers separated by a colon, meaning *hour*:*minute*. You can also append an *am* or *pm* suffix; otherwise the commands assume a 24-hour clock. The suffix *zulu* may be used to indicate GMT. The special names *noon*, *midnight*, *now*, and *next* are also recognized.

You can specify an optional *date* as either a month name followed by a day number (and possibly a year number preceded by an optional comma), or a day of the week (fully spelled or abbreviated to three characters). Two special "days", *today* and *tomorrow* are recognized. If you have not provided a *date*, *today* is assumed if the given hour is greater than the current hour and *tomorrow* is assumed if it is less. If the given month is less than the current month (and no year is given), **next year** is assumed.

The optional *increment* is a number suffixed by one of the following: *minutes*, *hours*, *days*, *weeks*, *months*, or *years*. (The singular form is also accepted.)

−r*job*	Remove jobs previously scheduled with **at**.
−l	[*job*] Report all jobs (by job number) scheduled for the invoking user.

EXAMPLES

at and **batch** read from standard input the commands to be executed at a later time. **sh**(1) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
batch
sort filename >outfile
<control-D> (hold down "CTRL" and press 'D')
```

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

```
batch <<!
sort filename 2>&1 >outfile | mail loginid
!
```

To have a job reschedule itself, invoke *at* from within the shell procedure, by including code similar to the following within the shell file:

```
echo "sh shellfile" | at 1900 thursday next week
```

Some examples of simple, yet valid **at** command lines are shown here:

```
at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at 5 pm Friday
```

FILES

/usr/lib/cron	Main cron directory
/usr/lib/cron/at.allow	List of allowed users
/usr/lib/cron/at.deny	List of denied users
/usr/lib/cron/queue	Scheduling information
/usr/spool/cron/atjobs	Spool area

DIAGNOSTICS

Complains about various syntax errors and times out of range.

SEE ALSO

kill(1), mail(1), nice(1), ps(1), sh(1), sort(1).
cron(1M) in *Managing SysV System Software*.

NAME

awk – pattern scanning and processing language

SYNOPSIS

awk [–Fc] [*prog*] [*parameters*] [*files*]

DESCRIPTION

awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as –f *file*. The *prog* string should be enclosed in single quotes (′) to protect it from the shell.

Parameters, in the form x=... y=... etc., may be passed to awk.

Files are read in order; if there are no files, the standard input is read. The file name – means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using FS; see below). The fields are denoted **$1**, **$2**, . . . ; **$0** refers to the entire line.

A pattern-action statement has the form:

```
pattern { action }
```

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] . . . }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next     # skip remaining patterns on this input line
exit     # skip the rest of the input
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the operators +, –, *, /, %, and concatenation (indicated by a blank). The C operators ++, ––, +=, –=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (").

The *print* statement prints its arguments on the standard output (or on a file if >*expr* is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its expression list according to the format [see **printf**(3S) in the *SysV Programmer's Reference*].

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp*, *log*, *sqrt*, and *int*. The last truncates its argument to an integer; *substr*(*s*, *m*, *n*) returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf*(*fmt*, *expr*, *expr*, ...) formats the expressions according to the **printf**(3S) format given by *fmt* and returns the resulting string.

Patterns are arbitrary Boolean combinations (!, | | , &&, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep* (see **grep**(1)). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

    expression matchop regular-expression
    expression relop expression

where: relop is any of the six relational operators in C, and a matchop is either (for *contains*) or ! (for *does not contain*). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character *c* may be used to separate the fields by starting the program with:

```
BEGIN { FS = c }
```

or by using the −F*c* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default new-line); and OFMT, the output format for numbers (default %.6g).

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

```
          { s += $1 }
END       { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }
       { print }
```

command line: **awk −f program n=5 input**

BUGS

Input white space is not preserved on output if fields are involved.
There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

SEE ALSO

grep(1), sed(1).
lex(1), printf(3S) in the *SysV Programmer's Reference*.

NAME

banner – make posters

SYNOPSIS

banner strings

DESCRIPTION

banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

SEE ALSO

echo(1).

NAME

basename, **dirname** – deliver portions of path names

SYNOPSIS

basename *string* [*suffix*]
dirname *string*

DESCRIPTION

basename deletes any prefix ending in / and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside substitution marks (‘ ‘) within shell procedures.

dirname delivers all but the last level of the path name in *string*.

EXAMPLES

The following example, invoked with the argument /usr/src/**cmd**/cat.c, compiles the named file and moves the output to a file named **cat** in the current directory:

```
cc $1
mv a.out ‘basename $1 '\.c'‘
```

The following example sets the shell variable **NAME** to /usr/src/**cmd**:

```
NAME=‘dirname /usr/src/cmd/cat.c‘
```

SEE ALSO

sh(1).

NAME

at, batch – execute commands at a later time

SYNOPSIS

at *time* [*date*] [+ *increment*]
at –r*job*...
at –l [*job* ...]

batch

DESCRIPTION

at and **batch** read commands from standard input to be executed at a later time. **at** allows you to specify when the commands should be executed, while jobs queued with **batch** execute when system load level permits.

Standard output and standard error output are mailed to you, unless you redirect them elsewhere. Shell environment variables, current directory, umask, and ulimit are retained when you execute either **at** or **batch**. Open file descriptors, traps, and priority are lost.

You can use **at** if your name appears in the file **/usr/lib/cron/at.allow**. If that file does not exist, the file **/usr/lib/cron/at.deny** determines whether or not you are allowed to use **at**. If neither file exists, only **root** can submit a job. The allow/deny files consist of one user name per line. These files can only be modified by the superuser.

batch submits a batch job. It is equivalent to the command **at now** with the exceptions that **batch** goes into a differenct queue and responds earlier with error messages.

OPTIONS

The following options apply to **at** only:

[*time*] [+ *increment*]
: Specify *time* when commands are to be executed. One- and two-digit numbers indicate hours, four-digit numbers show hours and minutes. You may alternately specify the time as two numbers separated by a colon, meaning *hour:minute*. You can also append an *am* or *pm* suffix; otherwise the commands assume a 24-hour clock. The suffix *zulu* may be used to indicate GMT. The special names *noon*, *midnight*, *now*, and *next* are also recognized.

You can specify an optional *date* as either a month name followed by a day number (and possibly a year number preceded by an optional comma), or a day of the week (fully spelled or abbreviated to three characters). Two special "days", *today* and *tomorrow* are recognized. If you have not provided a *date*, *today* is assumed if the given hour is greater than the current hour and *tomorrow* is assumed if it is less. If the given month is less than the current month (and no year is given), **next** year is assumed.

The optional *increment* is a number suffixed by one of the following: *minutes*, *hours*, *days*, *weeks*, *months*, or *years*. (The singular form is also accepted.)

−r*job*	Remove jobs previously scheduled with **at**.
−l	[*job*] Report all jobs (by job number) scheduled for the invoking user.

EXAMPLES

at and **batch** read from standard input the commands to be executed at a later time. sh(1) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
batch
sort filename >outfile
CTRL/D
```

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

```
batch <<!
sort filename 2>&1 >outfile | mail loginid
!
```

To have a job reschedule itself, invoke *at* from within the shell procedure, by including code similar to the following within the shell file:

```
echo "sh shellfile" | at 1900 thursday next week
```

Some examples of simple, yet valid **at** command lines are shown here:

```
at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at 5 pm Friday
```

FILES

/usr/lib/cron	Main cron directory
/usr/lib/cron/at.allow	List of allowed users
/usr/lib/cron/at.deny	List of denied users
/usr/lib/cron/queue	Scheduling information
/usr/spool/cron/atjobs	Spool area

DIAGNOSTICS

Complains about various syntax errors and times out of range.

SEE ALSO

kill(1), mail(1), nice(1), ps(1), sh(1), sort(1).
cron(1M) in *Managing SysV System Software*.

NAME

bc – arbitrary-precision arithmetic language

SYNOPSIS

bc [**–c**] [**–l**] [*file* . . .]

DESCRIPTION

bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The **bc**(1) utility is actually a preprocessor for **dc**(1), which it invokes automatically unless the **–c** option is present. In this case the **dc** input is sent to the standard output instead.

Th͏e value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or new-lines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of **dc**(1). Assignments to *ibase* or *obase* set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. ''Auto'' variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

PROGRAM SYNTAX

The syntax for **bc** programs is shown below; (L means letter a–z, E means expression, S means statement).

Comments

Enclosed in /* and */

Names

Simple variables: L
Array elements: L [E]
The words ''ibase'', ''obase'', and ''scale''

Other Operands

Arbitrarily long numbers with optional sign and decimal point
(E)
sqrt (E)

length (E)	Number of significant decimal digits
scale (E)	Number of digits right of decimal point

L (E , ... , E)

Operators

```
+  –  *  /  %  ^   (% is remainder; ^ is power)
++  —  (Prefix and postfix; apply to names)
==  <=  >=  !=  <  >
=  =+  =–  =*  =/ =%  =^
```

Statements
 E
 { S ; ... ; S }
 if (E) S
 while (E) S
 for (E ; E ; E) S
 null statement
 break
 quit

Function Definitions
 define L (L ,..., L) {
  auto L, ... , L
  S; ... S
  return (E)
 }

Functions in –l Math Library

s(x)	sine
c(x)	cosine
e(x)	exponential
l(x)	log
a(x)	arctangent
j(n,x)	Bessel function

All function arguments are passed by value.

OPTIONS

–c Compile only. The output is send to the standard output.

–l Argument stands for the name of an arbitrary precision math library.

EXAMPLE

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

defines a function to compute an approximate value of the exponential function and

```
        for(i=1; i<=10; i++) e(i)
```

prints approximate values of the exponential function of the first ten integers.

BUGS

The **bc** command does not yet recognize the logical operators, && and | |.
For statement must have all three expressions (E's).
Quit is interpreted when read, not when executed.

FILES

/usr/lib/lib.b	Mathematical library
/usr/bin/dc	Desk calculator proper

SEE ALSO

dc(1).

NAME

bdiff – big diff

SYNOPSIS

bdiff *file1 file2* [n] [–s]

DESCRIPTION

bdiff is used in a manner analogous to **diff**(1) to find which lines in two files must be changed to bring the files into agreement. Its purpose is to allow processing of files which are too large for **diff**.

OPTIONS

file1 (file2)	The name of a file to be used. If *file1 (file2)* is –, the standard input is read.
n	The number of line segments. The value of *n* is 3500 by default. If the optional third argument is given and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for **diff**, causing it to fail.
–s	Specifies that no diagnostics are to be printed by **bdiff** (silent option). Note, however, that this does not suppress possible diagnostic messages from **diff**(1), which **bdiff** calls.

bdiff ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes **diff** upon corresponding segments. If both optional arguments are specified, they must appear in the order indicated above.

The output of **bdiff** is exactly that of **diff**, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, **bdiff** does not necessarily find a smallest sufficient set of file differences.

FILES

/tmp/bd?????

DIAGNOSTICS

Use **help**(1) for explanations.

SEE ALSO

diff(1), help(1).

NAME

bfs – big file scanner

SYNOPSIS

bfs [–] name

DESCRIPTION

bfs is like **ed**(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes and 32K lines, with up to 512 characters, including new-line, per line (255 for 16-bit machines). **bfs** is usually more efficient than **ed**(1) for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where **csplit**(1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the w command. The optional – suppresses printing of sizes. Input is prompted with * if **P** and a carriage return are typed, as in **ed**(1). Prompting can be turned off again by inputting another **P** and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under **ed**(1) are supported. In addition, regular expressions may be surrounded with two symbols besides / and ?: > indicates downward search without wrap-around, and < indicates upward search without wrap-around. There is a slight difference in mark names: only the letters **a** through **z** may be used, and all 26 marks are remembered.

The **e**, **g**, **v**, **k**, **p**, **q**, **w**, =, ! and null commands operate as described under **ed**(1). Commands such as ——, +++–, +++=, **–12**, and **+4p** are accepted. Note that **1,10p** and **1,10** will both print the first ten lines. The **f** command only prints the name of the file being scanned; there is no *remembered* file name. The **w** command is independent of output diversion, truncation, or crunching (see the **xo**, **xt** and **xc** commands, below). The following additional commands are available:

xf *file*
: Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the **xf**. The **xf** commands may be nested to a depth of 10.

xn
: List the marks currently in use (marks are set by the **k** command).

xo [*file*]
: Further output from the **p** and null commands is diverted to the named *file*, which, if necessary, is created mode 666 (readable and writable by everyone), unless your **umask** setting (see **umask**(1)) dictates otherwise. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: *label*

This positions a *label* in a command file. The *label* is terminated by newline, and blanks between the : and the start of the *label* are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(. , .)xb/*regular expression*/*label*

A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between 1 and $.
2. The second address is less than the first.
3. The regular expression does not match at least one line in the specified range, including the first and last lines.

On success, **.** is set to the line matched and a jump is made to *label.* This command is the only one that does not issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

```
xb/^/ label
```

is an unconditional jump.
The xb command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe only a downward jump is possible.

xt *number*

Output from the p and null commands is truncated to at most *number* characters. The initial number is 255.

xv[*digit*] [*spaces*] [*value*]

The variable name is the specified *digit* following the xv. The commands **xv5100** or **xv5 100** both assign the value **100** to the variable **5**. The command **xv61,100p** assigns the value **1,100p** to the variable **6**. To reference a variable, put a % in front of the variable name. For example, using the above assignments for variables **5** and **6**:

```
1,%5p
1,%5
%6
```

will all print the first 100 lines.

```
g/%5/p
```

would globally search for the characters **100** and print each line containing a match. To escape the special meaning of %, a \ must precede it.

```
g/".*\%[cds]/p
```

could be used to match and list lines containing *printf* of characters, decimal integers, or strings.

Another feature of the **xv** command is that the first line of output from a UNIX system command can be stored into a variable. The only requirement is that the first character of *value* be an **!**. For example:

```
.w junk
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

would put the current line into variable **5**, print it, and increment the variable **6** by one. To escape the special meaning of **!** as the first character of *value*, precede it with a ****.

```
xv7\!date
```

stores the value **!date** into variable **7**.

xbz *label*

xbn *label*
These two commands will test the last saved *return code* from the execution of a UNIX system command (**!***command*) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string **size**.

```
xv55
: l
/size/
xv5!expr %5 – 1
!if 0%5 != 0 exit 2
xbn l
xv45
: l
/size/
xv4!expr %4 – 1
!if 0%4 = 0 exit 2
xbz l
```

xc [*switch*]
If *switch* is **1**, output from the **p** and null commands is crunched; if *switch* is **0** it is not. Without an argument, **xc** reverses *switch*. Initially *switch* is

set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

DIAGNOSTICS

? for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.

SEE ALSO

csplit(1), ed(1), umask(1).

NAME

bldt – display time operating system was built

SYNOPSIS

bldt [*options*] [*node_id*]

DESCRIPTION

bldt displays the time at which the running version of Domain/OS was built.

node_id (optional)	Display the build time of the node whose network root directory is *pathname*. Default if omitted: display build time of current node

OPTIONS

–n *node_spec* ...	Display build time of specified node[s].
–a	Display build time of all nodes.

EXAMPLES

$ **bldt //ward**

```
     **** Node 29C27.4B51 ****    "//ward"
Domain/OS kernel(3), revision 10.0, bl20.1 April 15, 1988  1:02:54 pm
```

$ **bldt –n //june**

```
     **** Node 29C27.CBB9 ****    "//june"
Domain/OS kernel(8), revision 10.0, bl17.3 February 9, 1988  8:12:37 am
```

$ **bldt –n CBB9**

```
     **** Node 29C27.CBB9 ****    "//june"
Domain/OS kernel(8), revision 10.0, bl17.3  \
                          February 9, 1988  8:12:37 am
```

NAME

cal – print calendar

SYNOPSIS

cal [[month] year]

DESCRIPTION

cal prints a calendar for a specified month and/or year. If neither is specified, **cal** prints a calendar for the present month only.

Both *year* and *month* must be Arabic numbers. The range for *year* is 1-9999. The range for *month* is 1-12.

EXAMPLES

To print a calendar for the entire year of 1988, type the following:

```
cal 1988
```

To print a calendar for December, 1988, type:

```
cal 12 1988
```

BUGS

The year always starts in January.

Note that ''cal 88'' refers to the year 88, not 1988.

NAME

calendar – reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

calendar provides an individual reminder sevice by consulting the file **calendar** in your login directory and printing out lines containing today's or tomorrow's date. You must create the file before **calendar** can successfully run.

A typical line in your **calendar** file may look like this:

> 12/15 Departmental meeting at 3 p.m.

calendar recognizes most month-day entries (e.g., 12/15, Dec. 15, december 15), but not day-month items (e.g., 15 December, 15/12). On weekends, "tomorrow" extends through Monday.

When an argument is present, **calendar** looks in all users' login directories for a file named **calendar** and sends any positive results by **mail**(1).

BUGS

Your calendar must be public information for you to get reminder service.
calendar's extended idea of ''tomorrow'' does not account for holidays.

FILES

/usr/lib/calprog to figure out today's and tomorrow's dates **/etc/passwd /tmp/cal***

SEE ALSO

mail(1).

NAME

lp, cancel – send/cancel requests to an LP line printer

SYNOPSIS

lp [–c] [–d*dest*] [–m] [–n*number*] [–o*option*] [–s] [–t*title*] [–w*files*]
cancel [ids] [printers]

DESCRIPTION

lp arranges for the named files and associated information (collectively called a "request") to be printed by a line printer. If no file names are mentioned, the standard input is assumed. A dash (–) used as a file name indicates the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

lp associates a unique "id" with each request and prints it on the standard output. This id can be used later to cancel (see **cancel**) or find the status (see **lpstat**(1)) of the request.

OPTIONS

The following options to **lp** may appear in any order and may be intermixed with file names:

–c Makes copies of the *file(s)* to be printed immediately when **lp** is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the –c option is not given, then you should be careful not to remove any of the *file(s)* before the request has been printed completely. Without the –c option, any changes made to the named *files* after the request is made, but before it is printed, will be reflected in the printed output.

–d*dest* Chooses *dest* as the printer or class of printers where printing will take place. If *dest* is a printer, the request will be printed only on that specific printer. If *dest* is a class of printers, the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer availability, file space limitation, etc.), requests for specific destinations may not be accepted (see **accept**(1M) and **lpstat**(1)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see **lpstat**(1)).

–m Sends mail after the *files* have been printed (see **mail**(1)). By default, no mail is sent.

–n*number* Prints *number* copies of the output (default is 1).

–o*option* Specifies a printer-dependent or class-dependent *option*. Several such *options* may be collected by specifying –o more than once. For more information about what are valid *options*, see **Models** in **lpadmin**(1M).

–s	Suppresses messages from **lp**(1) such as "request id is ...".
–t*title*	Prints *title* on the banner page of the output.
–w	Writes a message to your terminal after the *files* have been printed. If you are not logged in, mail is sent instead.

Cancel cancels line printer requests made by **lp**(1). The command line arguments can be either request *id*s (as returned by **lp**(1)) or *printer* names (for a complete list of *printer* names, use **lpstat**(1)). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which is currently printing on that *printer*. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1).
accept(1M), lpadmin(1M), lpsched(1M) in the *Managing SysV System Software*.

NAME

cat – concatenate and print files

SYNOPSIS

cat [**–u**] [–s] [–v [–t] [–e]] file ...

DESCRIPTION

cat reads each file in sequence and writes it on the standard output. If no input file is given, or if the argument "–" is encountered, **cat** reads from the standard input file.

OPTIONS

–u Produce unbuffered output. (The default is buffered output.)

–s Ignore non-existent files.

–v Make non-printing characters visible (except for tabs, new-lines and form-feeds). Print control characters (CTRL key and *X*) as "^X", the delete character (DELETE key - octal 0177) as a caret with a question mark "^?", and non-ASCII characters (with the high bit set) as M-*x*, where *x* is the character specified by the seven low-order bits.

–t With the –v option, print tabs as ^I.

–e With the –v option, print a dollar sign ($) at the end of each line (prior to the new-line).

The –t and –e options are ignored if the –v option is not specified.

EXAMPLES

To write *file1* on standard output, type the following:

```
# cat file1
```

To write standard input to *file1*, use this command:

```
# cat >file1
```

To write *file1* and *file2* to *file3*, type this:

```
# cat file1 file2 >file3
```

BUGS

Command formats such as **cat file1 file2 >file1** destroy the original data in *file1*. Be careful when using shell special characters.

SEE ALSO

cp(1), pg(1), pr(1).

NAME

cb – C program beautifier

SYNOPSIS

cb [**–s**] [**–j**] [**–l** leng] [file ...]

DESCRIPTION

cb reads C programs from its arguments or from the standard input, and writes them on the standard output with spacing and indentation displaying the structure of the code. Under default options, **cb** preserves all user new-lines.

OPTIONS

–s	Causes code to conform to the style of Kernighan and Ritchie in *The C Programming Language*.
–j	Puts split lines back together.
–l *leng*	Splits lines that are longer than *leng*.

BUG

Punctuation that is hidden in preprocessor statements causes indentation errors.

SEE ALSO

cc(1).
The C Programming Language. Prentice-Hall, 1978.

NAME

cc – C compiler

SYNOPSIS

cc [*options*] *files*

DESCRIPTION

cc is an interface to the preprocessor (**cpp**), the Domain C compiler, and the link editor (**ld**). cc processes the supplied *options* and then executes the various tools with the proper arguments. cc accepts several types of files as arguments:

Files whose names end with **.c** are taken to be C source programs and may be preprocessed, compiled, and link edited. The compilation process may be stopped after the completion of any pass if the appropriate *options* are supplied. If the compilation process runs through the compiler then an object program is produced and is left in the file whose name is that of the source with **.o** substituted for **.c**. However, the **.o** file is normally deleted if a single C program is compiled and then immediately link edited. Files whose names end in **.i** are taken to be preprocessed C source programs and may be compiled and link edited. Files whose names do not end in **.c**, or **.i** are handed to the link editor.

Assembly source programs (files whose names end in .s) are not supported.

Since cc usually creates files in the current directory during the compilation process, it is necessary to run cc in a directory in which a file can be created.

Not all standard UNIX *options* are available. Furthermore, some unique *options* are provided by the Domain cc command. If the cc command does not recognize an *option* as a preprocessor or compiler *option*, it assumes that it is an *option* for the link editor (**ld**) and passes it along. The *options* the cc interface recognizes as preprocessor *options* are: **–C**, **–D**, **–H**, **–I**, and **–U**. The link editor *options* are: **–a**, **–l**, **–L**, **–m**, **–M**, **–o**, **–r**, **–s**, **–t**, **–u**, **–V**, **–x** and **–z**. *Options* that are recognized but ignored are: **–ds**, **–dl**, **–f**, **–F** and **–S**. When you use these *options*, you get a warning that they are not supported.

OPTIONS

–A cpu,*id* Generates code for a particular class of processor. Legal values for *id* are:

any	Standard M68000 code
160	DSP160 code
460	DN460 code
660	660 code
90	DSP90 code
330	DN330 code
560	DN560 code
570	DN570 code
580	DN580 code
3000	DN3000 code

4000	DN4000 code
FPX	Floating-Point Accelerator Board
PEB	Performance Enhancement Board

–A nansi Does not compile with ANSI rules. This *option* passes *–ntype* to the compiler and does not define the preprocessor symbol __STDC__.

–A runtype,*type*
Passes *type* information to compiler and linker.

–A systype,*type*
Defines the target system type (*systype*) for the compiled object. *type* can be one of:

TYPE	DESCRIPTION
any	Version independent
bsd4.2	Berkeley version 4.2
bsd4.3	Berkeley version 4.3
sys5	UNIX System V
sys5.3	UNIX System V.3

This replaces the **–T** *option.*

–c Suppresses the linking phase of the compilation and forces an object file to be produced, even if only one program is compiled.

–E Runs only **cpp**(1) on the named C programs, and sends the result to the standard output.

–g Causes the compiler to generate additional information needed for using **dbx**(1) or **dde**(1).

–H (**cpp** switch) Prints out to *stderr* the pathname of each file included during this compilation.

–o *outfile* Produces an output object file named *outfile*. The name of the default file is **a.out**. This is a link editor *option.*

–O Turns on compilation phase optimitzations.

–p Produces code that counts the number of times each routine is called; also, automatically calls **monitor**(3C). Produces a **mon.out** file at normal terminal execution of the object program. An execution file is then generated by using **prof**(1).

–P Runs only **cpp**(1) on the named C programs, and leaves the result on corresponding files suffixed with **.i**. Passes this *option* to **cpp**(1).

–qg Produces profiled code that allows profiling with **gprof**(1). Produces a **gmon.out** file at normal termination of execution of the object program.

−qp Produces profiled code where the **p** argument produces identical results to the **−p** *option* (allows profiling with **prof**(1)).

−T *systype* Defines the target system type (*systype*) for the compiled object. *systype* can be one of:

any	Version independent
bsd4.2	Berkeley version 4.2
bsd4.3	Berkeley version 4.3
sys5	UNIX System V
sys5.3	UNIX System V.3

Note this *option* is identical to the −A **systype**,*type option*, but may become obsolete in a future release. We recommend using −A **systype**,*systype*.

−V Prints the version of the compiler and/or link editor that is invoked.

−W*c,arg1*,[*arg2...*]
Hands off the arguments *argi* to pass *c* where *c* is one of **p**, **0**, or **l**, indicating the preprocessor, compiler, or link editor, respectively. Using **−W0** enables you to use **/com/cc** *options* that are not available with **/bin**/cc. For example: **−W0, −pic** passes **−pic** to the compiler.

−Y[**p0lSILU**], *dir*
Specifies a new pathname, *dirname*, for the locations of the tools and directories designated by the first argument.

p	Preprocessor (**cpp**)
0	Compiler (cc)
l	Link editor (**ld**)
S	Directory containing start-up routine (/**usr/lib/crt0.o**)
I	Default include directory searched by preprocessor (/**usr/include**)
L	First default library directory searched by link editor (/**usr/lib**)
U	Second default library directory searched by link editor (no default)

If the location of a tool is being specified, the new pathname for the tool will be /**dirname/tool**. If more than one −Y *option* is applied to any one tool or directory, the last occurrence holds.

−B*string* **−t**[**p02al**] These *options* will be removed in the next release. Use the −Y *option*.

cc also recognizes −C, −D, −H, −I and −U and passes these *options* and their agruments directly to the preprocessor without using the −W *option*. Similarly, cc recognizes −a, −l, −L, −m, −M, −o, −r, −s, −t, −u, −V, −x, −z and passes these *options* and their arguments directly to the loader. See **cpp**(1) and **ld**(1).

Other agruments are taken to be C compatible object programs, typically produces by an earlier cc run, or perhaps libraries of C compatible routines and are passed directly to the link editor. These programs, together with the results of any compilations specified, are link edited (in order given) to produce an executable program with name **a.out** unless the **–o** *option* of the link editor is used.

If cc is put in file *prefixcc* the prefix will be parsed off the command and used to call the tools, i.e., *prefix*tool. For example, OLDcc will call OLDcpp, OLDcomp, OLDoptim, OLDas and OLDld and will link OLDcrt1.o. Therefore, one MUST be carefule when moving cc around. The prefix will apply to the preprocessor, compiler, link editor, and the start-up routines.

NOTES

By default, the return value from a compiled C program is completely random. The only two guaranteed ways to return a specific value are to explicitly call **exit**(2) or to leave the function **main**() with a ''*return expression;*'' construct.

FILES

file.c	C source file
file.i	Preprocessed C source file
file.o	object file
a.out	Link edited output
LIBDIR/crt0.o	Start-up routine
TMPDIR/*	Temporary files
LIBDIR/cpp	Preprocessor, **cpp**(1)
/usr/apollo/lib/cc	Compiler
BINDIR/ld	Link editor, **ld**(1)

LIBDIR is usually **/usr/lib**
BINDIR is usually **/bin**
TMPDIR is usually **/usr/tmp** but can be redefined by setting the environment variable **TMPDIR** [see **tempnam**() in **tmpnam**(3S)].

SEE ALSO

ld(1), cpp(1), gencc(1), lint(1), prof(1), dbx(1), tmpnam(3S).
Domain C Language Reference
"The C Programming Language", Kernighan, B.W. and Ritchie, D.M. Prentice-Hall, 1978.

NAME

cd – change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

If *directory* is not specified, **cd** uses the value of shell parameter **$HOME** as the new working directory. If *directory* specifies a complete path starting with a slash (/), a period (.), or two consectutive periods (..), *directory* becomes the new working directory. If neither case applies, **cd** tries to find the designated directory relative to one of the paths specified by the **$CDPATH** shell variable. **$CDPATH** has the same syntax as, and similar semantics to, the **$PATH** shell variable. **cd** must have execute (search) permission in *directory*.

Because a new process is created to execute each command, **cd** would be ineffective if it were written as a normal command; therefore, it is recognized by, and is internal to, the shell.

EXAMPLES

To change your working directory to the directory called *mydata*, type the following:

```
# cd mydata
```

To advance your working directory one level up in the naming hierarchy, use this command:

```
# cd ..
```

SEE ALSO

pwd(1), sh(1).
chdir(2) in the *SysV Programmer's Reference*.

NAME

cdc –change the delta commentary of an SCCS delta

SYNOPSIS

cdc –rSID [–m[mrlist]] [–y[comment]] files

DESCRIPTION

cdc changes the delta commentary for the SID (SCCS IDentification string) specified by the **–r** option, of each named SCCS file.

Delta commentary is defined to be the Modification Request (MR) and comment information normally specified via the **delta**(1) command (**–m** and –y arguments).

If a directory is named, **cdc** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read (see *BUGS*) and each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to **cdc**, which may appear in any order, consist of *options* and file names. All options described below apply independently to each named file.

OPTIONS

–r*SID* Specifies the SCCS *ID*entification (SID) string of a delta for which the delta commentary is to be changed.

–m*mrlist* Supplies a list of MR numbers to be added and/or deleted in the delta commentary of the SID specified by the **–r** option. The SCCS file must have the v flag set. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of **delta**(1). In order to delete an MR, precede the MR number with the character ! (see *EXAMPLES*). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a ''comment'' line. A list of all deleted MRs is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If **–m** is not used and the standard input is a terminal, the prompt MRs? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The MRs? prompt always precedes the comments? prompt (see –y option).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the v flag has a value [see **admin**(1)], it is taken to be the name of a program (or shell procedure) which validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, **cdc** terminates and the delta commentary remains unchanged.

–y*[comment]*	Arbitrary text used to replace the comments already existing for the delta specified by the **–r** option. The previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If –y is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

EXAMPLES

```
cdc –r1.6 –m"bl78-12345 !bl77-54321 bl79-00001" –ytrouble s.file
```

adds bl78-12345 and bl79-00001 to the MR list, removes bl77-54321 from the MR list, and adds the comment **trouble** to delta 1.6 of s.file.

```
cdc –r1.6 s.file
MRs? !bl77-54321 bl78-12345 bl79-00001
comments? trouble
```

does the same thing.

BUGS

If SCCS file names are supplied to the *cdc* command via the standard input (– on the command line), then the **–m** and –y keyletters must also be used.

To modify the delta commentary, you must be either (1) the creator of the delta, or (2) the owner of the SCCS file and directory.

FILES

x-file	[see **delta**(1)]
z-file	[see **delta**(1)]

DIAGNOSTICS

Use *help*(1) for explanations.

SEE ALSO

admin(1), delta(1), get(1), prs(1), sccsfile(4).
help(1) in the *Using Your SysV Environment.*

NAME

cflow – generate C flowgraph

SYNOPSIS

cflow [**–r**] [**–ix**] [**–i_**] [**–d***num*] *files*

DESCRIPTION

cflow analyzes a collection of C, yacc, lex, and object files and attempts to build a graph charting the external references. Files suffixed with **.y**, **.l**, and **.c** are yacced, lexed, and C-preprocessed as appropriate. The results of the preprocessed files, and files suffixed with **.i**, are then run through the first pass of **lint**(1). Files suffixed with **.o**, have information extracted from their symbol tables. The results are collected and turned into a graph of external references, which is displayed on the standard output.

Each line of output begins with a reference number, followed by a suitable number of tabs indicating the level, then the name of the global symbol followed by a colon and its definition. Usually only function names that do not begin with an underscore are listed (see the **-i** *options* below). For information extracted from C source, the definition consists of an abstract type declaration (**char**, for instance), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (e.g., *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only < > is printed.

OPTIONS

In addition to the **–D**, **–I**, and **–U** *options* [which are interpreted just as they are by **cc**(1) and **cpp**(1)], the following *options* are interpreted by **cflow**:

–r	Reverses the "caller:callee" relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.
–ix	Includes external and static data symbols. The default is to include only functions in the flowgraph.
–i_	Includes names that begin with an underscore. The default is to exclude these functions (and data if *–ix* is used).
–dnum	The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be ignored.

EXAMPLE

As an example, given the following in **file.c**:

```
int	i;

main()
{
	f();
	g();
	f();
}

f()
{
	i = h();
}
```

the command

cflow –ix file.c

produces the output

```
1	main: int(), <file.c 4>
2		f: int(), <file.c 11>
3			h: <>
4			i: int, <file.c 1>
5		g: <>
```

When the nesting level becomes too deep, the output of **cflow** can be piped to **pr**(1), using the **–e** *option*, to compress the tab expansion to something less than every eight spaces.

DIAGNOSTICS

Notifies you of bad *options*. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C-preprocessor).

BUGS

Files produced by **lex**(1) and **yacc**(1) cause the reordering of line number declarations which can confuse **cflow**. To get proper results, feed **cflow** the **yacc** or **lex** input.

SEE ALSO

cc(1), cpp(1), lex(1), lint(1), nm(1), yacc(1).
pr(1) in *Using Your SysV Environment.*

NAME

chacl – change access control list

SYNOPSIS

chacl [–odfvLR] *<spec> file...*

chacl [–odfvLR] –D *<sid> file ...*

chacl [–odfvLR] [–u *<owner>* **] [–g** *<group>* **] [–z** *<organization>*] *file...*

chacl [–odfvLR] { –c | –l | –n } *file ...*

chacl [-vLR] { –B | –S } *file...*

DESCRIPTION

The **chacl** command changes the entries in an object's access control list (ACL). Use the specification (*spec*) part of the command line either to set the rights for a given subject identifier (*sid*), or to change the inheritance mechanisms of a directory. The specification syntax, shown below, is similar to **chmod**'s symbolic mode form.

<spec>:	*<sid><op><rights>* | *<req><op><inh>* | *<spec>*[,*<spec>*...]
<sid>:	*%.%.%* | *<req>*
<req>:	[ugzo] | a
<op>:	= | + | –
<inh>:	[UP]
<rights>:	[prwxksI]

OPTIONS

–B

–S The –**B** (BSD) and –S (SysV) options simply set a directory to use the appropriate semantics. Any existing ACLs are removed, and the protections on the directory are determined by the current **umask**(2). Owner, group, and organization inheritance are determined using the appropriate semantics (SysV, all from current process; BSD, owner from current process, group from directory. Organization is marked "ignore" for both).

–c Force calculation of the extended entry mask. The mask represents the maximum rights of all extended ACL entries, and is automatically calculated each time **chacl** is run. This option is used to undo the effects of the **chmod** command, as **chmod** affects the mask as well as the world required entry (*%.%.%*) when changing rights for "other".

–l Set local access. With local access set, an object can be accessed only from the node on which it is located.

–n Set network access.

–o Make the changes on the ACL itself for the objects specified. If the –o, **–d**, or –f options are not specified, –o is assumed These options can be used in any combination.

–d	Make the changes on the initial directory ACL.
–f	Apply the changes to the initial file ACL.
–v	(verbose) List each destination as the ACL is changed.
–L	Follow any soft links encountered, and operate on the object to which the link points. Since soft links in Domain/OS do not have ACLs, attempting to change a soft link without the –L flag simply results in a warning, with no change.
–R	Apply the changes recursively to any directories encountered among the files listed. Be very careful when combining this option with the –L option!
–D	Delete extended entries from an ACL. Required entries may not be deleted, so <*sid*> must be an actual subject identifier (see below).
–u	Set the owner field in an ACL.
–g	Set the group field in an ACL.
–z	Set the organization field in an ACL.

SUBJECT IDENTIFIERS

The *sid* (Subject IDentifier) used in the first form (<*sid*><*op*><*rights*>) is a way of specifying a user or set of users. It may include a username, group name and organization name, any of which may be replaced with the wildcard %, or left off, as described in **acl**(7). The special cases **u**, g, z and o refer to the required entries in the ACL for user, group, organization and world. The special case a refers to the all of the above (user, group, organization and world), as does a null SID field. These special cases do not affect required entries that are marked "Ignore". Short user IDs that are a combination of the letters **u**, g, z and o are distinguished from the special cases by the use of the % syntax described above. Thus oz+x adds execute rights for other and organization, whereas oz.%.%+x or oz..+x adds execute rights for just the user oz.

ACCESS RIGHTS

Access rights are specified by the *op* (operator) and *rights* parameters to **chacl**. Valid operators are =, +, and –. The = operator specifies absolute rights for the SID. If the ACL already contains an entry for this SID, **acl** changes it to contain the rights listed. Otherwise, it adds an entry with the specified SID and access rights.

If you specify the + operator, the rights are added to any existing rights for the specified SID. Likewise, the – operator removes the rights from the ACL entry for the SID. If no entry exists for the SID, the entire ACL is searched for more general entries that apply to this SID. The specified rights are then added to or removed from this aggregate set of rights, and a new entry is created for the specific SID.

Access rights consist of any combination of the following letters:

Files:

p	Protect rights; allow rights to be changed
r	Read rights; allows file to be read
w	Write rights; allows file to be written
x	Execute rights: allows file to be executed
k	Keep; prevents file from being deleted or having its name changed
s	Set ID; usable only with **u**, g and z (user, group, and organization); causes this executable to be run with the effective ID of the user, group or organization

Directories:

p	Protect rights; allow rights to be changed
r	Read rights; allows directory to be listed
w	Write rights; allows names to be added, changed or deleted
x	Execute rights; allows subordinate objects to be used, without allowing the directory to be listed; also called search rights
k	Keep; prevents directory from being deleted or having its name changed

The following is used alone, and overrides any other rights specified:

I	Ignore; used to ignore the rights in the required owner, group, organization, and other entries

To change the inheritance properties of a directory, use the second form of ACL specification (*<req><op><inh>*). In this case, the first field must consist only of required entries **u**, g, z, or a (user, group, organization, all) and the second field specifies the inheritance option.

The valid inheritance options are as follows:

U	Umask; the rights for new objects in this directory are those requested by the process creating them as modified by the **umask**(2) of that process
P	Process; inherit user, group or organization from the process creating a new object in this directory

EXAMPLES

chacl g+w *

Add write rights for the group to each file in this directory.

chacl a=rx foo

Give owner, group, organization and world read and execute rights to the file **foo**.

chacl ugz=I .

Ignore the required entries for owner, group and organization.

chacl %.os=prwx .

Give the os group full rights to this directory.

chacl ..mktg–pw,..r_d=prwx .

Be sure that the **mktg** organization does not have write or protect rights and that **r_d** has full rights to the current directory.

chacl –B /usr/u/bar

Strip any extended ACLs from **/usr/u/bar**, and set it up as a BSD directory.

chacl –D arnold.staff *

Delete any ACL entries referring to **arnold.staff**.

chacl –od susan+x /usr/u/zap

Always allow **susan** to use objects in **/usr/u/zap** directory, and to search any new sub-directories.

chacl –odf user= magicdir

Insure that **user** has no rights to **magicdir**, nor to any files or sub-directories subsequently created in **magicdir**.

chacl –f ugz=UP .

Newly created files in the current directory inherit owner, group, and organization (and the associated rights) from the process.

chacl –df g-P .

Do not inherit group from the process, that is, inherit it from this directory for new files and sub-directories.

SEE ALSO

lsacl(1), cpacl(1), chmod(1), chgrp(1), chorg(1),chown(2), umask(2), salacl(1M), acl(5)

NAME

chfn, chsh, passwd – change password file information

SYNOPSIS

passwd [–s] [–f] [*name* **]**
chsh *shell*
chfn

DESCRIPTION

The **passwd** command changes or installs a password, log-in shell (–s option), or GECOS information field (–f option) associated with the user *name* (your own name by default).

chsh changes a log-in shell, and is equivalent to **passwd –s**.

chfn changes the GECOS information field, and is equivalent to **passwd –f**.

When altering a password, **passwd** prompts for the current password and then for the new one; you must supply both. You must type the new password twice to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet, and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the super-user can change a password; owners must prove they know the old password.

When altering a log-in shell, (using **passwd –s** or **chsh**) the program displays the current log-in shell and then prompts for the new one. The new log-in shell must be one of the approved shells listed in **/etc/shells** unless you are the super-user. If **/etc/shells** does not exist, the only shells that can be specified are **/bin/sh**, **/bin/csh**, **/bin/ksh**, and **/com/sh**.

The super-user can change anyone's log-in shell; normal users can only change their own log-in shell(s).

When altering the GECOS information field, (using **passwd –f** or **chfn**), the program displays the current information, broken into fields, as interpreted by the **finger**(1) program (among others) and prompts for new values. These fields can include a user's "real life" name, office room number, office phone number, and home phone number. Each prompt includes a default value, which is enclosed between brackets. The default value is accepted simply by typing a carriage return. To enter a blank field, the word "none" can be typed. Phone numbers can be entered with or without hyphens. It is a good idea to run **finger** after changing the GECOS information to make sure everything is set up properly.

The super-user can change anyone's GECOS information; normal users can only change their own.

EXAMPLE

Below is a sample run:

```
% passwd -f
Name [Biff Studsworth II]:
Room number (Exs: 597E or 197C) []: 521E
Office Phone (Ex: 1632) []: 1863
Home Phone (Ex: 987532) [5771546]: none
```

NOTES

On Domain/OS systems, the /etc/passwd file is a typed file, which is automatically generated by the registry daemon. The registry administrator can make the person information in the registry read-only, in which case normal users cannot change the "Name" field.

FILES

/etc/passwd	The file containing all of this information
/etc/shells	The list of approved shells

SEE ALSO

login(1), finger(1), passwd(4), crypt(3C), edrgy(1M);
Using Your SysV Environment

NAME

chown, chgrp – change owner or group

SYNOPSIS

chown owner file ...

chown owner directory ...

chgrp group file ...

chgrp group directory ...

DESCRIPTION

chown changes the owner of the *files* or *directories* to *owner*. The owner may be either a decimal user ID or a login name found in the password file.

chgrp changes the group ID of the *files* or *directories* to *group*. The group may be either a decimal group ID or a group name found in the group file.

If either command is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

Only the owner of a file (or the super-user) may change the owner or group of that file.

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the ls –l command leads you to believe.

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1).
chown(2), group(4), passwd(4) in the *SysV Programmer's Reference*.

NAME

chmod – change mode

SYNOPSIS

chmod *mode file ...*

chmod *mode directory ...*

DESCRIPTION

chmod allows the permissions of the named *files* or *directories* to be changed according to **mode**, which may be absolute or symbolic. An absolute mode is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
20#0	set group ID on execution if # is **7**, **5**, **3**, or **1**
	enable mandatory locking if # is **6**, **4**, **2**, or **0**
1000	sticky bit (sticky bit is not supported in SysV)
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

Symbolic changes are stated using letters that correspond both to access classes and to the individual permissions themselves. Permissions to a file may vary depending on your user identification number (UID) or group identification number (GID). Permissions are described in three sequences each having three characters:

User	Group	Other
rwx	rwx	rwx

In this example, user, group, and others all have reading, writing, and execution permission to a given file. There are two categories for granting permissions: the access class (who) and the permissions themselves. Thus, to change the mode of a file's (or directory's) permissions using **chmod**'s symbolic method, use the following syntax:

[*who*] *operator* [*permission(s)*], ...

A command line using the symbolic method would appear as follows:

chmod g+rw *file*

This command would make *file* readable and writable by the group.

who is a combination of the letters **u** for owner's permissions), g (group), and o (other). The letter a stands for **ugo**, the default if *who* is omitted.

Operator can be plus (+) to add *permission* to the file's mode, minus (-) to take away *permission*, or equal (=) to assign *permission* absolutely (reset all other bits).

Permission is any compatible combination of the following letters:

r	reading permission
w	writing permission
x	execution permission
s	user or group set-ID is turned on
t	sticky bit (sticky bit is not supported in SysV)
l	mandatory locking will occur during access

Multiple symbolic modes separated by commas can be given, though no spaces may appear between these modes. Operations are performed in the order given. Multiple symbolic letters following a single operator cause the corresponding operations to be performed simultaneously. The letter s is only useful with **u** or g, and **t** only works with **u.**

EXAMPLES

chmod a–x *file*

chmod 444 *file*

The first examples deny execution permission to all. The absolute (octal) example permits only reading permissions.

chmod go+rw *file*

chmod 606 *file*

These examples make a file readable and writable by the group and others.

chmod +l *file*

This causes a file to be locked during access.

chmod =rwx,g+s *file*

chmod 2777 *file*

These last two examples enable all to read, write, and execute the file; and they turn on the set group-ID.

BUGS

Mandatory file and record locking (l) refers to a file's ability to have its reading or writing permissions locked while a program is accessing that file. It is not possible to permit group execution and enable a file to be locked on execution at the same time. In addition, it is not possible to turn on the set-group-ID and enable a file to be locked on execution at the same time. The following examples,

chmod g+x,+l *file*

chmod g+s,+l *file*

are, therefore, illegal usages and will elicit error messages.

Only the owner of a file or directory (or the super-user) may change a file's mode. Only the super-user may set the sticky bit. In order to turn on a file's set-group-ID, your own group ID must correspond to the file's, and group execution must be set.

The DOMAIN system's single-level store requires that all files be mappable and, therefore, readable by the OS. This means that SysV does not recognize execute-only or write-only files. For example, if you type **chmod 111 foo**, SysV automatically sets read permissions for the owner as follows:

```
-r-xr-xr-x  1 owner     unix           5 May 22 11:47 fc
```

Also, if you type **chmod 222 foo**, SysV automatically sets read permissions for owner as follows:

```
-rw-rw-rw-  1 harper   sys           5 May 22 11:50 foo
```

Only the owner of a file (or the super-user) may change its mode.

To set the group ID, the group associated with the file must correspond to your current group ID.

SEE ALSO

ls(1).
chmod(2) in the *SysV Programmer's Reference*.

NAME

chown, chgrp – change owner or group

SYNOPSIS

chown owner file ...

chown owner directory ...

chgrp group file ...

chgrp group directory ...

DESCRIPTION

chown changes the owner of the *files* or *directories* to *owner*. The owner may be either a decimal user ID or a login name found in the password file.

chgrp changes the group ID of the *files* or *directories* to *group*. The group may be either a decimal group ID or a group name found in the group file.

If either command is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

Only the owner of a file (or the super-user) may change the owner or group of that file.

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the ls –l command leads you to believe. For more information see the "Mapping Remote Users" section of Chapter 10 of the *SysV System Administrator's Guide*.

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1).
chown(2), group(4), passwd(4) in the *SysV Programmer's Reference*.

NAME

chfn, chsh, passwd – change password file information

SYNOPSIS

passwd [**–s**] [**–f**] [*name*]
chsh *shell*
chfn

DESCRIPTION

The **passwd** command changes or installs a password, log-in shell (–s option), or GECOS information field (–f option) associated with the user *name* (your own name by default).

chsh changes a log-in shell, and is equivalent to **passwd –s**.

chfn changes the GECOS information field, and is equivalent to **passwd –f**.

When altering a password, **passwd** prompts for the current password and then for the new one; you must supply both. You must type the new password twice to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet, and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the super-user can change a password; owners must prove they know the old password.

When altering a log-in shell, (using **passwd –s** or **chsh**) the program displays the current log-in shell and then prompts for the new one. The new log-in shell must be one of the approved shells listed in /etc/**shells** unless you are the super-user. If /etc/**shells** does not exist, the only shells that can be specified are **/bin/sh**, **/bin/csh**, **/bin/ksh**, and **/com/sh**.

The super-user can change anyone's log-in shell; normal users can only change their own log-in shell(s).

When altering the GECOS information field, (using **passwd –f** or **chfn**), the program displays the current information, broken into fields, as interpreted by the **finger**(1) program (among others) and prompts for new values. These fields can include a user's "real life" name, office room number, office phone number, and home phone number. Each prompt includes a default value, which is enclosed between brackets. The default value is accepted simply by typing a carriage return. To enter a blank field, the word "none" can be typed. Phone numbers can be entered with or without hyphens. It is a good idea to run **finger** after changing the GECOS information to make sure everything is set up properly.

The super-user can change anyone's GECOS information; normal users can only change their own.

EXAMPLE

Below is a sample run:

```
% passwd -f
Name [Biff Studsworth II]:
Room number (Exs: 597E or 197C) []: 521E
Office Phone (Ex: 1632) []: 1863
Home Phone (Ex: 987532) [5771546]: none
```

NOTES

On Domain/OS systems, the **/etc/passwd** file is a typed file, which is automatically generated by the registry daemon. The registry administrator can make the person information in the registry read-only, in which case normal users cannot change the "Name" field.

FILES

/etc/passwd	The file containing all of this information
/etc/shells	The list of approved shells

SEE ALSO

login(1), finger(1), passwd(4), crypt(3C), edrgy(1M);
Using Your SysV Environment

NAME

cmp – compare two files

SYNOPSIS

cmp [–l] [–s] file1 file2

DESCRIPTION

cmp compares two files. (If *file1* is –, the standard input is used.) Under default options, **cmp** makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

OPTIONS

–l	Prints the byte number (decimal) and the differing bytes (octal) for each difference.
–s	Prints nothing for differing files; return codes only.

DIAGNOSTICS

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

SEE ALSO

comm(1), diff(1).

NAME

col – filter reverse line feeds

SYNOPSIS

col [**–bfh**]

DESCRIPTION

col reads the standard input and writes the standard output. It performs the line overlays implied by reverse line feeds (ESC-7 in ASCII) and by forward and reverse half line feeds (ESC-9 and ESC-8). **col** is particularly useful for filtering multicolumn output made with the **.rt** command of **nroff**(1) and output resulting from using the **tbl**(1) preprocessor.

Although **col** accepts half line motions in its input, it normally does not emit them on output. Instead, it moves text that would appear between lines to the next lower full line boundary.

The control characters SO (ASCII code 017), and SI (016) are assumed to start and end text in an alternate character set. **col** remembers the character set (primary or alternate) associated with each printing character read. On output, **col** generates SO and SI characters where necessary to maintain the correct treatment of each character.

All control characters are removed from the input except space, backspace, tab, return, newline, and ESC (033) followed by one of 7, 8, 9, SI, SO, and VT (013). This last character is an alternate form of full reverse line feed, for compatibility with some other hardware conventions. All other non-printing characters are ignored.

OPTIONS

–b	Assumes that the output device in use is not capable of backspacing. If several characters are to appear in the same place, only the last one read will be taken.
–f	Allows the output to contain half-line feeds (ESC-9). Even with this option it will never contain either kind of reverse line motion.
–h	Converts white space to tabs to shorten printing time.

BUGS

col can't back up more than 128 lines.
There can be no more than 800 characters, including backspaces, on a line.

SEE ALSO

troff(1), tbl(1)

NAME

comb – combine SCCS deltas

SYNOPSIS

comb files

DESCRIPTION

comb generates a shell procedure [see **sh**(1)] which, when run, reconstructs the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, **comb** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored. The generated shell procedure is written on the standard output.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file. each **get –e** generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the **–o** keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file. This argument causes **comb** to generate a shell procedure which, when run, produces a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

100 * (original – combined) / original

It is recommended that before any SCCS files are actually combined, one should use this option to determine exactly how much space is saved by the combining process. *S*CCS *ID*entification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file. A *list* (see **get**(1) for the syntax of a *list*) of deltas to be preserved. All other deltas are discarded.

If no keyletter arguments are specified, **comb** preserves only leaf deltas and the minimal number of ancestors needed to preserve the tree.

BUGS

comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

FILES

s.COMB	The name of the reconstructed SCCS file.
comb?????	Temporary.

Use *help*(1) for explanations.

SEE ALSO

admin(1), delta(1), get(1), prs(1), sccsfile(4).

NAME

comm – select or reject lines common to two sorted files

SYNOPSIS

comm [– [**123**]] file1 file2

DESCRIPTION

comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see **sort**(1)), and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name – means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus **comm –12** prints only the lines common to the two files; **comm –23** prints only lines in the first file but not in the second; **comm –123** prints nothing.

SEE ALSO

cmp(1), diff(1), sort(1), uniq(1).

NAME

cp – copy files

SYNOPSIS

cp [**–CcfiopPsv**] *file1 file2*

cp [**–CcfiopPrsv**] *file ... directory*

DESCRIPTION

cp copies *file1* onto *file2*. By default, **cp** preserves the mode and owner of *file2* if *file2* already exists; otherwise it uses the mode of the source file modified by the current **umask**(2) is used.

In the second form, one or more *files* are copied into the *directory* with their original filenames.

cp refuses to copy a file onto itself.

OPTIONS

–i	Prompt the user with the filename whenever the copy will cause an old file to be overwritten. An answer of 'y' causes **cp** to continue. Any other answer prevents it from overwriting the file.
–p	Attempt to preserve (duplicate) in copies the modification times and modes of the source files, ignoring the present **umask**.
–r	If any of the source files is a directory, copy each subtree rooted at that name; in this case the destination must be a directory.

Domain/OS SysV OPTIONS

–C	Change the names of any existing files which would have been overwritten. The current date is appended to the filename, in the format *@mm.dd*[*.n*]. If this name already exists, an additional number is appended. If you specify the –C option, the files copied in will not adopt the mode and owner of the existing files.
–c	Change the names of any existing files which would have been overwritten. The current date is appended to the filename, in the format *.mm.dd*[*.n*]. If this name already exists, an additional number is appended. If you specify the –c option, the files copied in do not adopt the mode and owner of the existing files.
–f	Force locked files to be overwritten, and mark the overwritten files to be deleted when they become unlocked. If you specify the **–f** option, the files copied in do not adopt the mode and owner of the existing files.
–o	Copy each file as a typed object, without attempting to open a stream to the file. This is useful in cases where opening a stream would succeed, but not yield the entirety of the underlying object.

−s	Treat symbolic links as files to be copied, rather than copying the destination of the link. This is especially useful when the **−r** option is used to copy an entire directory tree that may contain links to other file systems.
−v	Print the name of each file copied, on the standard output (verbose).
−P	Attempt to preserve any extended access control list (ACL) on the source files, ignoring both the present **umask** and the destination ACL.

SEE ALSO

cat(1), cpacl(1), mv(1), rcp(1C), acl(7)

NAME

cpacl – copy access control list

SYNOPSIS

cpacl [–odfitvLR] *source destinations ...*

DESCRIPTION

cpacl copies access control lists (ACLs). If you do not specify an option, **cpacl** assumes **–o**.

The first argument specifies the object from which to copy the ACL. This ACL is then applied to the remaining files and directories as specified. Since only directories have fields specifying ACL inheritance, use of the **–d**, **–f**, and **–i** options requires that the source object be a directory. The destinations can be anything, and the appropriate set of fields are applied (see below).

OPTIONS

–o Copies the object ACL associated with *source* to each of *destinations*. This is the default if you do not specify an option.

–d Copies the initial directory ACL associated with the specified directory to all *destinations* that are directories.

–f Copies the initial file ACL associated with the specified directory to all *destinations* that are directories.

–i Copies an initial default ACL onto an object ACL. The source must be a directory; destinations that are files receive the initial file ACL of the source, and directories w receive the initial directory ACL of the source. If you use **-i** in combination with the **-d** or **-f** options, **cpacl** copies the initial directory or initial file ACL in the usual way.

–t Uses the object ACL of the source as a template for the initial file or directory ACL of the destination. If the source is a file,**cpacl** copies its ACL to the initial file ACL of any directories. If the source is a directory, **cpacl** copies its ACL only to the initial directory ACL of target directories.

–v Produce verbose output; that is, show the name of each target as an ACL is copied to it.

–L Directs **cpacl** to follow any soft links encountered, and operate on the object to which the link points. Since soft links in Domain/OS do not have ACLs, attempting to copy to or from a soft link without the –L flag produces just a warning, with no change.

–R Recursively descends any directories encountered among the destination objects. Be very careful when you combine this option with the –L option!

EXAMPLES

To copy the ACL on foo to objects **bar** and **zap**, enter the following:

```
cpacl foo bar zap
```

To copy the ACL, initial directory ACL, and initial file ACL on the /usr/u/**fred** directory to all objects in the current directory (files will have only the object ACL applied), enter the following:

```
cpacl –odf /usr/u/fred *
```

To copy the initial file ACL on **fred** to all files in that directory, and the initial directory ACL on **fred** to all sub-directories (also, copy the initial file and directory ACLs to the initial file and directory ACLs of any subdirectories), enter the following:

```
cpacl –idf /usr/u/fred /usr/u/fred/*
```

To copy the object ACL on **file-template** to the initial file ACL of the current directory, enter the following:

```
cpacl –t /usr/u/fred/file-template
```

To copy the object ACL on **dir-template** to the initial directory acl of the current directory, enter the following:

```
cpacl –t /usr/u/fred/dir-template
```

SEE ALSO

cp(1), chacl(1), lsacl(1), dbacl(1), chmod(1), chgrp(1), chorg(1), chown(1), acl(5), umask(2), salacl(1M)

NAME

cpio – copy file archives in and out

SYNOPSIS

cpio –o[acBv]

cpio –i[BcdmrtuvfsSb6] [patterns]

cpio –p[adlmuv] directory

DESCRIPTION

cpio –o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. Output is padded to a 512-byte boundary.

cpio –i (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio –o**. Only files with names that match *patterns* are selected. *Patterns* are regular expressions given in the name-generating notation of **sh**(1). In *patterns*, meta-characters ?, *, and [...] match the slash / character. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (i.e., select all files). Each *pattern* should be surrounded by double quotes. The extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files will be those of the previous **cpio –o**. The owner and group of the files will be that of the current user unless the user is super-user, which causes **cpio** to retain the owner and group of the files of the previous **cpio –o**.

cpio –p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

OPTIONS

a	Resets *access* times of input files after they have been copied. Access times are not reset for linked files when **cpio -pla** is specified.
B	Blocks input/output 5,120 bytes to the record. (Does not apply to the *pass* option; meaningful only with data directed to or from a character special device, e.g. /dev/rmt/0m.)
d	Creates *directories* as needed.
c	Writes header information in ASCII *character* form for portability. Use this option when origin and destination machines are different types.
r	Interactively *renames* files. If you type a null line, the file is skipped. (Not available with **cpio -p**.)
t	Prints a *table of contents* of the input. No files are created.
u	Copies *unconditionally* (normally, an older file will not replace a newer file with the same name).

v (verbose)	Causes a list of file names to be printed. When used with the **t** option, the table of contents looks like the output of an **ls –l** command (see ls(1)).
l	*Link*s files rather than copying them. Usable only with the **–p** option.
m	Retains previous file *modification* time. This option is ineffective on directories that are being copied.
f	Copies in all *files* except those in *patterns*.
s	*Swap*s bytes within each half word. Use only with the **–i** option.
S	*Swap*s halfwords within each word. Use only with the **–i** option.
b	Reverses the order of the *bytes* within each word. Use only with the **–i** option.
6	Processes an old (i.e. UNIX System *Sixth* Edition format) file. Only useful with **–i** (copy in).

NOTES

If **cpio –i** tries to create a file that already exists and the existing file is the same age or newer, **cpio** will output a warning message and not replace the file. (The **-u** option can be used to unconditionally overwrite the existing file.)

cpio assumes four-byte words.

If **cpio** reaches end of medium (end of a diskette for example), when writing to (-o) or reading from (-i) a character special device, **cpio** will print the message:

```
If you want to go on, type device/file name when ready.
```

To continue, you must replace the medium and type the character special device name (**/dev/rdiskette** for example) and carriage return. You may want to continue by directing **cpio** to use a different device. For example, if you have two floppy drives you may want to switch between them so **cpio** can proceed while you are changing the floppies. (A carriage return alone causes the **cpio** process to exit.)

Path names are restricted to 256 characters.

Only the super-user can copy special files.

Blocks are reported in 512-byte quantities.

EXAMPLES

The following examples show three uses of **cpio**.

When standard input is directed through a pipe to **cpio –o**, it groups the files so they can be directed (>) to a single file (*../newfile*). Instead of "ls," you could use find, echo, cat, etc. to pipe a list of names to cpio. You could direct the output to a device instead of a file.

ls | **cpio –o** >*../newfile*

cpio –i uses the output file of **cpio –o** (directed through a pipe with cat in the example), takes out those files that match the patterns (*memo/a1*, *memo/b**), creates directories below the current directory as needed (-d option), and places the files in the appropriate directories. If no patterns were given, all files from "newfile" would be placed in the directory.

cat newfile | **cpio –id** *memo/a1 memo/b**

cpio –p takes the file names piped to it and copies or links (-l option) those files to another directory on your machine (*newdir* in the example). The -d options says to create directories as needed. The –m option says retain the modification time. (It is important to use the -depth option of find to generate path names for **cpio**. This eliminates problems **cpio** could have trying to create files under read-only directories.)

find . –depth –print | **cpio –pdlmv** *newdir*

SEE ALSO

ar(1), find(1), ls(1), tar(1).
cpio(4) in the *SysV Programmer's Reference*.

NAME

cpp – the C language preprocessor

SYNOPSIS

LIBDIR/cpp [*option* . . .] [*ifile* [*ofile*]]

DESCRIPTION

cpp is invoked as the first pass of any C compilation by the **cc**(1) command. Therefore, **cpp**'s output is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, **cpp** and the rest of the C compilation package are modified to follow these changes.

cpp optionally accepts two file names as arguments. *ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

OPTIONS

–P Preprocesses the input without producing the line control information used by the next pass of the C compiler.

–C Passes along C-style comments (except those found on **cpp** directive lines). By default, **cpp** strips out C-style comments.

–U*name*

Removes any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. Following is the current list of these possibly reserved symbols. On Apollo computers, unix, apollo, and aegis are defined.

operating system:	unix, dmert, gcos, ibm, os, tss, aegis
hardware:	interdata, pdp11, u370, u3b, u3b5, u3b2, u3b20d, vax, apollo
UNIX system variant:	RES, RT
lint(1):	lint

–D*name*

–D*name*=*def*

Defines *name* with value *def* as if by a **#define**. If no =*def* is given, *name* is defined with value 1. The **–D** option has lower precedence than the **–U** option. That is, if the same name is used in both a **–U** option and a **–D** option, the name will be undefined regardless of the order of the options.

–T Uses only the first eight characters to distinguish preprocessor symbols. This option is included for backward compatibility.

–I*dir* Changes the algorithm for searching for **#include** files whose names do not begin with a slash (/) to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in double quotes (" ") are searched for first in the directory of the file with the **#include** line, then in directories named in **–I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in angle brackets (<>), the directory of the file with the **#include** line is not searched.

–Y*dir* Uses directory *dir* in place of the standard list of directories when searching for #**include** files.

–H Prints, one per line on standard error, the path names of included files.

–t*sys* Sets the environment variable SYSTYPE to *sys* while **cpp** is running. This *option* is useful for setting the resolution of systype-dependent links. For example, if your systype is sys5.3 and you specify **–tbsd4.3**, the file **/usr/include/rdmb.h** resolves to **/bsd4.3/usr/include/ndmb.h** instead of **/sys5.3/usr/include/ndmb.h**.

Two special names are understood by **cpp**. The name __**LINE**__ is defined as the current line number (as a decimal integer) as known by **cpp**, and __**FILE**__ is defined as the current file name (as a C string) as known by **cpp**. They can be used anywhere (including in macros) just as any other defined name.

DIRECTIVES

All **cpp** directive lines start with # in column 1. Any number of blanks and tabs is allowed between the # and the directive. The directives are:

#define *name token-string*
Replace subsequent instances of *name* with *token-string*.

#define *name(arg, ..., arg) token-string*
Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma-separated sets of tokens, and a) followed by *token-string*, where each occurrence of an *arg* in the *token-string* is replaced by the corresponding set of tokens in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, **cpp** re-starts its scan for names to expand at the beginning of the newly created *token-string*.

#undef *name*
Cause the definition of *name* (if any) to be forgotten from now on. No additional tokens are permitted on the directive line after *name*.

#ident *"string"*
Put *string* into the .comment section of an object file.

#include *"filename"*
#include *<filename>*
Include at this point the contents of *filename* (which will then be run through **cpp**). When the *<filename>* notation is used, *filename* is only searched for in the standard places. See the –**I** and –Y options above for more detail. No additional tokens are permitted on the directive line after the final " or >.

#line *integer-constant "filename"*

Causes **cpp** to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file from which it comes. If *"filename"* is not given, the current file name is unchanged. No additional tokens are permitted on the directive line after the optional *filename*.

#endif

Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**. No additional tokens are permitted on the directive line.

#ifdef *name*

The following lines appear in the output only if *name* has been the subject of a previous **#define** and not the subject of an intervening **#undef**. No additional tokens are permitted on the directive line after *name*.

#ifndef *name*

The following lines appear in the output only if *name* has not been the subject of a previous **#define**. No additional tokens are permitted on the directive line after *name*.

#if *constant-expression*

The following lines appear in the output only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the ?: operator, the unary –, !, and ~ operators are all legal in *constant-expression*. Operator precedence is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by **cpp** should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

To test whether either of two symbols, *foo* and *fum*, are defined, use

```
#if defined(foo) || defined(fum)
```

#elif *constant-expression*

An arbitrary number of **#elif** directives is allowed between a **#if**, **#ifdef**, or **#ifndef** directive and a **#else** or **#endif** directive. The lines following **#elif** appear in the output only if the preceding test directive evaluates to zero, all intervening **#elif** directives evaluate to zero, and the *constant-expression* evaluates to non-zero. If *constant-expression* evaluates to non-zero, all succeeding **#elif** and **#else** directives will be ignored. Any *constant-expression* allowed in a **#if** directive is allowed in a **#elif** directive.

#else The following lines appear in the output only if the preceding test directive evaluates to zero, and all intervening **#elif** directives evaluate to zero. No additional tokens are permitted on the directive line.

The test directives and the possible #else directives can be nested. In addition, the following directives are recognized by **cpp** and passed to the Domain C compiler:

```
# apollo
# apollo_bit
# debug
# eject
# list
# module
# nolist
# systype
# backstop
# section
# inhibit
# attribute
# options
```

NOTES

Using **cpp** other than through the **cc**(1) command is not suggested. See **m4**(1) for a general macro processor.

The unsupported **−W** option enables the **#class** directive. If it encounters a #class directive, **cpp** exits with code 27 after finishing all other processing. This option provides support for ''C with classes''.

Because the standard directory for included files may be different in different environments, use the following form of the **#include** directive:

#include <file.h>

rather than one with an absolute path, like:

#include "/usr/include/file.h"

cpp warns about the use of the absolute pathname.

FILES

INCDIR	Standard directory list for **#include** files, usually /usr/include
LIBDIR	/usr/lib

DIAGNOSTICS

The error messages produced by **cpp** are intended to be self-explanatory. The file name and line number where the error occurred are printed along with the diagnostic.

SEE ALSO

cc(1), lint(1), m4(1). *Domain C Language Reference.*

NAME

cpscr – copy the current display to a file

SYNOPSIS

cpscr [–inv] [–append] [–gpr[_bitmap]] *pathname*

DESCRIPTION

cpscr copies the current screen image, without clearing it, to the file you specify. Use the **prf** (print_file) command to print the file.

Use the DM command **cpo** to copy the screen without creating a new process window or changing the current transcript pad. **cpo** invokes the **cpscr** command from the DM without creating a pad or window. Thus, press <CMD> and type

```
cpo /usr/apollo/bin/cpscr pathname
```

You may copy small portions of a black and white screen (such as a single window) with the DM command **xi**.

By default, black and white screens are copied into a GMF file. Color screens are copied into a GPR bitmap.

pathname (required)	Specify file to that the screen is copied.

OPTIONS

–inv	Invert image. Use this option to store the image in reverse video. Black screen pixels become white and white screen pixels become black. Do not used this option with the **–gpr_bitmap** option or on color nodes.
–append	Appends a black and white screen image to an existing GMF file. You cannot use this option with the **–gpr_bitmap** option or on color nodes.
–gpr_bitmap	Use this option to copy a black and white screen into a GPR bitmap file rather than a GMF file. This option has no meaning for color nodes since color screens are already copied into GPR bitmaps.

EXAMPLES

Invert and copy the current screen image to the specified file. Since the command line is echoed in the shell's process transcript pad prior to execution, this command will appear in the resulting image.

```
$ cpscr –inv //us/looky_there
```

```
<cmd>
Command: cpo /usr/apollo/bin/cpscr -inv //us/looky_there
```

Same result as in the previos example, but the **cpscr** line will not appear in the plotted output.

NAME

crddf – create, display, or modify a device descriptor file

SYNOPSIS

crddf [*options* ...] *pathname*

DESCRIPTION

crddf creates, displays, or modifies a device descriptor file (DDF). A DDF defines a peripheral bus unit (PBU) device for which you have written a driver. See *Writing Device Drivers with GPIO Calls* for details on both DDFs and PBUs.

crddf is valid only if the general purpose input/output (GPIO) software is running on your network.

pathname (required) Specify name of the DDF to be created, modified, or displayed.

OPTIONS

–
: Reads further options from standard input. Signal completion with **–end**.

–at
: Specifies that device lives on the AT-compatible bus.

–call_library *pathname*
: Specifies pathname of the call side of the device driver library. This option is required.

–check
: Checks the DDF to ensure that all required fields have been specified.

–cleanup_routine [*entry_name*]
: Specifies the entry-point name of the clean-up routine in the call library. Omitting the entry name deletes a previously existing clean-up routine.

–csr_offset *port_number*
: Specifies the offset into the control status register (CSR) page, in hexadecimal format, at which the device's control/status registers are located. Device drivers may use this information during controller initialization.

–csr_page *iova*
: Specifies the hexadecimal address of the CSR page for the device in the bus address space. The following information applies to the particular bus structure implemented on your system:

- Multibus: optional
- VME bus: optional. If specified, must be page-aligned and in the range C000-D000.

- AT-compatible bus: If specified, may indicate a range (for example, –csr_page 200 21F). If the second parameter is missing, a range of 8 consecutive bytes is assumed (for example, –csr_page 200 assumes a range of 200-207).

–debug Sets a flag that can be used to turn on debugging logic in a driver.

–display Displays the current contents of the DDF.

–dma_channel *channel-number*
Specifies to the driver the DMA channel number used by AT-compatible device. This is a Version 3 option.

–end Closes the updated DDF and exit.

–initialization_routine *entry_name* (required)
Specifies the entry-point name of the initialization routine in the call library.

–interrupt_library *pathname*
Specifies the pathname of the interrupt side of the device driver library.

–interrupt_routine level [*entry_name*] (required)
Specifies a level at which the device interrupts and the entry-point name of an optional interrupt routine.

–major *ddevice_number*
Specifies the DDF's major device number in range 0-31.

–minor *ddevice_number*
Specifies the DDF's minor device number in range 0-511.

–memory_base *iova* Specifies the MULTIBUS address that marks the base of a controller's local memory. If the specified *iova* is less than 64K this is a Version 2 option, if *iova* is greater than 64K, this is a Version 3 option.

–memory_size *length*
Specifies the size, in hexadecimal format, of the controller memory. If the specified *iova* less than 64K, this is a Version 2 option; if greater than 64K, this is a Version 3 option.

–multiple Specifies that the device driver supports more than one device and cause the **crddf** command to check the driver entry-point names listed in the DDF for each device to ensure that it doesn't load multiple copies of the same driver.

–node[f] {*node number*|*} (required)
: Specifies the hexadecimal node ID of the node to which the device is physically connected. **–nodef** suppresses the check which makes certain the node exists. You may use an asterisk (*) instead of the node ID to indicate the local node.

–quit
: Exits without modifying the original DDF.

–remddf *//node name*
: Specifies a remote node on which the DDF resides.

–replace
: Replaces (i.e., overwrite) an existing DDF with a new version. To modify only selected portions of an existing DDF, use **–update**.

–revision [*string*]
: Specifies an optional revision number as an 8-character string.

–serial_number [*string*]
: Specifies an optional serial number as a 16-character string.

–share
: Specifies a DDF for a controller that can be shared among multiple processes.

–stack_size [*decimal number*]
: Specifies the number of bytes, in decimal, to be allocated to the interrupt stack (default is 1024).

–type *type name*
: Specifies the DDF's type. The type must already be installed on the node.

–unit *unit number* (required)
: Specifies the unit number of the device (must be equal to the lowest interrupt level on which the device interrupts).

• MULTIBUS:	Must be in range 0-5.
• VME bus:	Must be in range 8-14.
• AT-compatible bus:	Must be in range 0-15.

–update
: Modifies selected portions of an existing DDF. If this option is specified, it must precede all other options on the command line. To replace a DDF completely, use **–replace**.

–user_info [*string*]
: Specifies up to 64 characters of optional user information (no embedded blanks).

–vme
: Specifies that device lives on VME bus. This is a Version 3 option.

–20_bit_addressing
: Specifies 20-bit memory address size of controller. You must use PBU2 calls.

EXAMPLES

1. Create a new DDF specifying only the required information.

```
$ crddf /dev/mt0 –
  New DDF.
  > -unit 3
  > -node 2F
  > -csr_page 1400
  > -call_library /lib/mt.lib
  > -initialization_routine mt_$init
  > -interrupt_library /lib/mt.int.lib
  > -interrupt_routine 3 mt_$int
  > -check
  No missing fields.
  > -end
$
```

2. Display a DDF.

```
$ crddf /dev/mt0 –display
  ddf version:   1
  device uid:    00030003 0000002F  (unit 3, node 2F)
  csr page iova: 1400
  call library:                /lib/mt.lib
  interrupt library:           /lib/mt.int.lib
  initialization entry point: mt_$init
  cleanup entry point:         mt_$cleanup
  interrupt stack size: 1024
  interrupt routines:
     level 0: [unused]
     level 1: [unused]
     level 2: [unused]
     level 3: mt_$int
     level 4: [unused]
     level 5: [unused]
     level 6: [unused]
     level 7: [unused]
     serial number:
     revision:
     user info:
$
```

3. Change the name of the interrupt routine in an existing DDF.

   $ **crddf /dev/mt0 –update –interrupt_routine 3 mt_$sio**

4. Replace a DDF on the node //**grip** with a new version.

```
$ crddf –remddf //grip /dev/x25 –
  > -replace
  > -unit 2
  > -node *
  > -call_library /sys/x25/x25_driver.lib
  > -interrupt_library /sys/x25/x25_driver_int.lib
  > -initialization_routine x25_driver_$init
  > -cleanup_routine x25_driver_$cleanup
  > -interrupt_routine 2 x25_driver_$int
  > -memory_base 7000
  > -memory_size 1000
  > -revision 7.0
  > -serial_number
  > -user_info release
  > -display
  > -end
$
```

5. Create a new DDF for a device that will be accessed through streams for the installed type **foodev**:

```
$ crddf /dev/foodev –
  New DDF.
  > -unit 3
  > -node *
  > -csr_page 1400
  > -call_library /lib/foodev.lib
  > -initialization_routine foodev_$init
  > -interrupt_library /lib/foodev.int.lib
  > -interrupt_routine 3 mt_$int
  > -type foodev
  > -check
  No missing fields.
  > -end
$
```

NAME

crontab – user crontab file

SYNOPSIS

crontab [*file*]
crontab –r
crontab –l

DESCRIPTION

crontab copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs.

You can use **crontab** if your name appears in the file **/usr/lib/cron/cron.allow**. If that file does not exist, **crontab** checks the file **/usr/lib/cron/cron.deny** to determine if you should be allowed access. If neither file exists, only root is allowed to submit a job. If **cron.allow** does not exist and **cron.deny** exists but is empty, global usage is permitted. The allow/deny files contain one user name per line.

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

minute (0–59),
hour (0–23),
day of the month (1–31),
month of the year (1–12),
day of the week (0–6 with 0=Sunday).

Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 * 1 would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, 0 0 * * 1 would run a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by \) is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your **$HOME** directory with an **arg0** of **sh**. Users who desire to have their *.profile* executed must explicitly do so in the crontab file. *Cron* supplies a default environment for every shell, defining **HOME**, **LOGNAME**, **SHELL(=/bin/sh)**, and **PATH(=:/bin:/usr/bin:/usr/lbin)**.

If you do not redirect the standard output and standard error of your commands, any generated output or errors is mailed to you.

OPTIONS

–r	Removes your **crontab** from the **crontab** directory.
–l	Lists the **crontab** file for the invoking user.

WARNING

If you inadvertently enter the **crontab** command with no argument(s), do not try to get out with a CTRL/D. This causes all entries in your **crontab** file to be removed. Instead, exit with a DEL.

FILES

/usr/lib/cron	Main cron directory
/usr/spool/cron/crontabs	Spool area
/usr/lib/cron/log	accounting Information
/usr/lib/cron/cron.allow	List of allowed users
/usr/lib/cron/cron.deny	List of denied users

SEE ALSO

sh(1).
cron(1M) in the *Managing Your SysV System Software*.

NAME

crp – create a process on a remote node

SYNOPSIS

crp –on *node_spec* [*options*] [*command line*]

DESCRIPTION

crp creates a process on a remote node.

command line (optional)
: Specify command line to be executed by the remote process. If the command string contains embedded blanks, enclose it in quotation marks.

OPTIONS

The following option, which specifies the remote node, is required:

–on *node_spec*
: Specify the remote node on which the process is to be created.

You can specify one of the following options.

–cp (default)
: Create a remote process running with standard streams connected to the current window. The option is not valid if **–cpo** or **–cps** is specified.

–nwp
: Do not create a window-pane legend indicating that the local window is connected to a remote process. Use with the **–cp** option only.

–cpo
: Create a remote process without a connection to the current window, and an identity of **user.none.none**. This option is not valid if **–cp** or **–cps** is specified. To stop these processes, you must first create a visible remote process running the shell, then issue the **sigp** command to stop the background process.

–cps
: Create a remote process without a connection to the current window, and an identity of **user.server.none**. This option is not valid if **–cp** or **–cpo** is specified. To stop these processes, you must first create a visible remote process running the shell, then issue the **sigp** command to stop the background process.

–n *name*
: Specify the name of the remote process. If this option is not specified, the default is **user id.node_id**. This allows remote processes to be traced to their originator.

–login *name* [*password*]
: Specify the log-in sequence for the remote process on the command line. If the password is omitted, the system prompts you for it. A null (zero-length) password is specified by the null string "".

Normally **–login** appears with **–cp**. However, you may use **–login** with **–cpo** and **–cps** as well. If **–login** is specified with either **–cpo** or **–cps**, the identity of the created process is the same as that of the caller (as opposed to **user.none.none** or **user.server.none**, respectively). This means that **–cpo** and **–cps** are identical if **–login** is also specified.

If you use **–login** with **–cpo** or **–cps**, you must place both the name and the password on the command line. No prompting is available in this case.

–me Specified instead of **–login**. If **–me** is specified, the created process on the remote node inherits the caller's working directory, naming directory, home directory text string, and SID. This is similar to popping up another shell except that the process is running on another node. If **–me** is specified with either **–cpo** or **–cps**, the identity of the created process is also that of the caller's (as opposed to **user.none.none** or **user.server.none**, respectively). This means that **–cpo** and **–cps** are identical if **–me** is also specified.

–quiet Suppress connection/disconnection messages in the transcript pad.

EXAMPLES

Create a process on node 532 running the shell, and login with the user ID joe.

```
$ crp –on 532 –login joe
```

Create a process on node aef running the shell, and inherit the current process state information.

```
$ crp –on 0aef –me
```

NAME

crpad – create a transcript pad and window

SYNOPSIS

crpad [*options*] [*pathname*]

DESCRIPTION

crpad creates a transcript pad, copies a file (or standard input) into that pad, and then opens a window into the pad. This new pad is not related to the transcript pad attached to processes running the shell; it is for viewing file contents only. This is primarily useful for displaying output being produced inside a pipeline without interrupting the flow of control in the pipe.

You cannot edit transcript pads. If you wish to place a file in a pad for editing, use the EDIT key or the DM command ce.

crpad –input behaves differently. This creates an edit pad and lets you create whatever text you want. When you close the edit pad (with wc or the EXIT key), that text is copied to standard output.

pathname (optional)	Specify the file to be copied into the pad. Not valid if **–input** is used. Default if omitted: copy standard input

OPTIONS

–i[nput]	Copies data from a temporary edit window to standard output. Not valid if **–tee** or **–pn** are specified.
–p[n] *pathname*	Specify a pathname for the pad. If you specify a pathname, the pad is saved in that file. Note that you can also save the pad after it is created by using the DM command **pn** (pad_name).
–t[ee]	Copy output to standard output in addition to the new pad.

EXAMPLES

Create a pad that displays the file **test.data**.

```
$ crpad test.data
```

Display the intermediate results in a pipeline.

```
$ $grep 256- phone.book | crpad -tee | sort >phone.book.local
```

Create an edit pad. When the pad is closed, sort the text edited and display it in a transcript pad.

```
$ crpad -input | sort | crpad
```

NAME

crty – create a new type

SYNOPSIS

crty [*options*] *type_name*

DESCRIPTION

crty creates a new type. It creates an identifier for the new type, and associates it with the supplied type name. New types are used to identify a new kind of manager for streams.

type_name (required) Specify the name to assign to the created type.

OPTIONS

–n *node_spec*	Specify the node on which the type is to be created. You may also specify the entry directory of a volume mounted for software installation, as shown in the example below. If this option is omitted, the type is created on the current node.
–l	List the type name/type identifier pair that is created.
–b[**inary**] *pathname*	Create the type from the specified object module (which was created by **crtyobj**). This allows you to use an object module (shipped on media like floppies, magnetic tapes, etc) to add a new type to a system.
–u *high.low*	Create the type with the specified unique identifier (UID). Give the high and low addresses for the UID as indicated. Note: Use this option only for system debugging. Misuse of this option may cause programs to behave incorrectly.

EXAMPLES

```
$ crty example_type -l
"example_type" 24BF9F41.100001FB created.

$ crty example_type -n //test_vol -l
"example_type" 24BFA6F8.200001FB created on volume //test_vol.
```

In the following example, the disk has been mounted for software installation. The disk's top level directory (cataloged as **/mount_disk** by the **mount**(1M) command) must contain a sys directory. If it does not, you get a "type manager directory not found" error.

```
$ crty example_type -n /mount_disk -l
"example_type" 24BFB71E.200001FB
 created on volume //my_node/mount_disk.
```

SEE ALSO

dlty(1), inty(1), lty(1), mount(1M)

NAME

crtyobj – create a type object module for binding

SYNOPSIS

crtyobj [*options*] *type_name* [*variable_name*]

DESCRIPTION

crtyobj creates an object module that contains a global symbol with the type UID. This module is bound with type managers. The variable is passed into calls to **trait_$mgr_dcl** to declare support for the specified type.

type_name (required)	Specify the name of the type for which an object module is to be created.
variable_name (optional)	Specify the variable name for the type UID. Default if omitted: name the variable *type_name_$uid*

OPTIONS

–b *bin_name*	Specify the output binary file name. The default is *type_name*.**bin.**
–sect *section_name*	Specify the section name for the data area in which the variable is declared. The default section name is **.data**.
–u *high.low*	Specify the type UID explicitly with the high and low addresses in the positions indicated. NOTE: Use this option only for system debugging.

EXAMPLES

```
$ crtyobj example_type example_$uid
$ bind –b example_mgr example_main.bin example_calls.bin example_type.bin
```

SEE ALSO

crty(1), dlty(1), lty(1)

NAME

csh – a shell (command interpreter) with C-like syntax

SYNOPSIS

csh [**–cefinstvVxX**] [**–D***name=value*] [*arg* ...]

DESCRIPTION

csh is a command-language interpreter that incorporates a history mechanism (see History Substitutions), job control facilities (see Jobs), interactive filename and username completion (see Filename Completion), and a C-like syntax.

csh begins by executing commands from the **.cshrc** file in your home directory. If this is a log-in shell, then it also executes commands from your **.login** file. CRT users often place an **stty crt** command in their **.login** files, and also invoke **tset**(1) there. To be able to use its job control facilities, users of **csh** must (and automatically) use the tty driver fully described in **tty**(4). This tty driver allows you to generate interrupt characters from the keyboard to tell jobs to stop. See **stty**(1) for details on setting options in the tty driver.

Normally, the shell then begins reading commands from the terminal, prompting with "% " . Upon reading a line of command input, the shell breaks it into *words*, places this sequence of words on the command-history list, parses it, and then executes each command in the line.

When a log-in shell terminates, it executes commands from the **.logout** file in your home directory.

LEXICAL STRUCTURE

Usually, **csh** splits input lines into words at blanks and tabs. The following, however, are exceptions to this:

- The characters: &, |, ;, >, <, (, and) form separate words. If doubled in &&, ||, <<, >>, these pairs form single words. You can make these parser metacharacters part of other words or prevent their special meaning by preceding them with a \ (backslash) character. A newline preceded by a backslash is equivalent to a blank.
- Strings enclosed in matched pairs of quotation marks, ´, `, or ", form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. Within pairs of ´ or " characters, a newline preceded by a backslash gives a true newline character.
- When the shell's input is not a terminal, the # character introduces a comment that continues to the end of the input line. To prevent this special meaning, you can precede the # by a \ or place it in quotations, using `, ´, and ".

COMMANDS

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by | characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by ; characters, and are then executed sequentially. A sequence of pipelines can be placed in the background by adding an & character at the end.

Any of the above may be placed in parentheses to form a simple command (which may be a component of a pipeline, and so on). You can also separate pipelines with || or && characters to indicate, as in the C language, that the second is to be executed only if the first fails or succeeds, respectively (see Expressions).

JOBS

csh associates a job with each pipeline. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers. When a job is started asynchronously with an &, **csh** prints a line similar to the following:

```
[1] 1234
```

This indicates that the job, which was started asynchronously, was job number 1 and had one (top-level) process whose process ID was 1234.

To suspend a running job, you must send it a stop signal, usually with CTRL/Z. Once **csh** has indicated that the job has been stopped (and has printed a prompt), you can manipulate the state of this job. You can put it in the background with the **bg** command, or run some other commands and then eventually bring the job back into the foreground with the **fg** command. A suspend takes effect immediately, causing **csh** to discard pending output and unread input. There is another special key, **CTRL/Y**, which does not generate a STOP signal until a program attempts to **read**(2) it. Type **CTRL/Y** ahead when you have prepared some commands for a job that you wish to stop after the program has read them. **CTRL/Y** is not supported in the Display Manager.

A job being run in the background stops if it tries to read from the terminal. Background jobs are normally allowed to produce output, but you can disable this by specifying the **stty tostop** command. Specifying **stty tostop** causes background jobs to stop when they try to produce output just as they do when they read input.

There are several ways to refer to jobs in the shell. The % character introduces a job name. Job number 1, for example, becomes **%1**. Naming a job brings it to the foreground; thus, **%1** is a synonym for **fg %1**, bringing job 1 back into the foreground. Similarly, specifying **%1 &** resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them, if the prefix is unambiguous. For example, %ex normally restarts a suspended ex(1) job, if there is only one suspended job whose name begins with the string "ex". You can also specify *%?string*, to indicate a job whose text contains *string*, if there is only one such job.

csh maintains a notion of the current and previous jobs. In output pertaining to jobs, it marks the current job with a + and the previous job with a –. The abbreviation %+ refers to the current job, and %– refers to the previous job. For close analogy with the syntax of the **history** mechanism (described below), a %% also represents the current job.

STATUS REPORTING

csh knows immediately when the state of a process changes. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. (This is so that it does not otherwise disturb your work.) However, if you set the **notify** shell variable, **csh** immediately reports status changes in background jobs. The **notify** shell command also marks a single process so that its status changes are immediately reported. By default, **notify** marks the current process. Thus, you only have to type **notify** after starting a background job to mark it.

When you try to leave the shell while jobs are stopped, you are warned that ''You have stopped jobs.'' You can use the **jobs** command to see which jobs are stopped. A second attempt to exit causes the suspended jobs to terminate without warning.

FILENAME COMPLETION

When the filename completion feature is enabled by setting the shell variable **filec** (see **set**), **csh** interactively completes filenames and usernames from unique prefixes, when they are input from the terminal followed by the escape character (the escape key, or CTRL/L). For example, if the current directory looks like

```
DSC.OLD    bin         cmd       lib       xmpl.c
DSC.NEW    chaosnet    cmtest    mail      xmpl.o
bench      class       dev       mbox      xmpl.out
```

and the input is

```
% vi ch<ESC>
```

csh completes the prefix ''ch'' to the only matching filename ''chaosnet'', changing the input line to

```
% vi chaosnet
```

However, if you specify

```
% vi D<ESC>
```

csh expands the input only to

```
% vi DSC.
```

and sounds the terminal bell to indicate that the expansion is incomplete, because two filenames match the prefix ''DSC''.

If a partial filename is followed by the end-of-file character (usually CTRL/D), then, instead of completing the name, **csh** lists all filenames matching the prefix. For example, the input

```
% vi D<CTRL/D>
```

causes all files beginning with "D" to be listed:

```
                DSC.NEW    DSC.OLD
```

while the input line remains unchanged.

You can use the same system of escape and end-of-file to expand partial usernames, if the word to be completed (or listed) begins with the character "~". For example, typing

```
                cd ~ro<CTRL/D>
```

may produce the expansion

```
                cd ~root
```

Set the variable **nobeep**, to inhibit the use of the terminal bell to signal errors or multiple matches.

Normally, all files in the particular directory are candidates for name completion. Files with certain suffixes can be excluded from consideration by setting the variable **fignore** to the list of suffixes to be ignored.

Thus, if you set **fignore** by the command

```
                % set fignore = (.o .out)
```

then typing

```
                % vi x<ESC>
```

results in the completion to

```
                % vi xmpl.c
```

ignoring the files **xmpl.o** and **xmpl.out**. However, if the only completion possible requires not ignoring these suffixes, they are not ignored. Also, **fignore** does not affect the listing of filenames by CTRL/D. All files are listed regardless of their suffixes.

HISTORY SUBSTITUTIONS

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in a previous command with little typing and much confidence. History substitutions begin with the character ! and can start anywhere in the input stream (providing that they do not nest). Precede the ! with a \ to prevent its special meaning. For convenience, a ! is passed unchanged when it is followed by a blank, tab, newline, =, or (. History substitutions also occur when an input line begins with a ^ (see below). Before being executed, input lines containing history substitution are echoed on the terminal as they could have been typed without history substitution.

csh saves input commands consisting of one or more words on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. The size of the history list is controlled by the **history** variable. The previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

For example, consider the following output from the **history** command:

```
     9  write michael
    10  ex write.c
    11  cat oldwrite.c
    12  diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but you can make the current event number part of the prompt by placing an ! in the **prompt** string.

Supposing the current event is 13, you can refer to previous events by event number, as in **!11** for event 11; relatively, as in !–2 for event 11; by a prefix of a command word, as in **!d** for event 12 or **!wri** for event 9; or by a string contained in a word in the command, as in **!?mic?**, also referring to event 9.

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case, !! refer to the previous command. Thus, !! alone is essentially a redo.

To select words from an event, you can follow the event specification by a : and a designator for the desired words. The words of an input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, and so on. The basic word designators are:

0	First (command) word
n	*n* 'th argument
^	First argument, i.e. '1'
$	Last argument
%	Word matched by (immediately preceding) ?*s*? search
x–*y*	Range of words
–*y*	Abbreviates '0–*y*'
*	Abbreviates '^–$', or nothing if only 1 word in event
*x**	Abbreviates '*x*–$'
x–	Like '*x**' but omitting word '$'

The : separating the event specification from the word designator can be omitted if the argument selector begins with a ^, $, * – or %. After the optional word designator can be placed a sequence of modifiers, each preceded by a :. The following modifiers are defined:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing '.*xxx*' component, leaving the root name.
e	Remove all but the extension '.*xxx*' part.
s/*l*/*r*/	Substitute *l* for *r*
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g. 'g&'.

p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like **q**, but break into words at blanks, tabs and newlines.

Unless preceded by a **g**, the modification is applied only to the first modifiable word. With substitutions, it is an error for no word to be applicable.

The left-hand side of substitutions are not regular expressions in the sense that they are in the editors (**ed**, **vi**, and so on) rather they are strings. You can use any character as the delimiter in place of a /. A \ quotes the delimiter into the *l* and *r* strings. The character & on the right-hand side is replaced by the text from the left. A \ also quotes an &. A null *l* uses the previous string either from an *l* or from a contextual scan string *s* in !?*s*?. The trailing delimiter in the substitution, as well as the trailing ? in a contextual scan, can be omitted if a newline follows immediately.

You can specify a history reference without an event specification, for example, **!$**. In this case, the reference is to the previous command unless a previous history reference occurred on the same line (in which case this form repeats the previous reference).

Thus, !?foo?^ !$' gives the first and last arguments from the command matching ?foo?.

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a ^. This is equivalent to !:s^ and provides a convenient shorthand for substitutions on the text of the previous line. Thus, **^lb^lib** fixes the misspelling of **lib** in the previous command. Finally, a history substitution may be surrounded with { **and** } if necessary, to insulate it from the characters that follow. Thus, after **ls –ld ~paul** you might type !{l}a to do **ls –ld ~paula** while !la looks for a command starting with **la.**

QUOTATIONS WITH SINGLE AND DOUBLE QUOTES

Placing strings in single and double quotation marks prevents all or some of the remaining substitutions. Those enclosed in single quotation marks are prevented any further interpretation; those in double quotation marks may be expanded as described below.

In both cases, the resulting text becomes all or part of a single word. In only one special case (see COMMAND SUBSTITUTION below) does a double-quoted string yield parts of more than one word; single-quoted strings never do.

ALIAS SUBSTITUTION

csh maintains a list of aliases that can be established, displayed, and modified by the **alias** and **unalias** commands. After it scans a command line, **csh** parses the line into distinct commands and checks the first word of each command, left-to-right, for an alias. If it finds one, it rereads the text that is the alias for that command (with the history mechanism available) as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, **csh** leaves the argument list unchanged.

For example, if the alias for **ls** is **ls –l**, the command **ls** /usr maps to **ls –l** /usr and the argument list is undisturbed. Similarly, if the alias for **lookup** is **grep** !^ /etc/passwd, then **lookup bill** maps to **grep bill** /etc/passwd.

Every time **csh** finds an alias, it transforms the input text and begins the aliasing process again on the reformed input line. Prevent looping (if the first word of the new text is the same as the old) by flagging the first word to prevent further aliasing. **csh** detects other loops and returns an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus, you can specify **alias print ´pr \!* | lpr´** to make a command that **pr**'s its arguments to the line printer.

VARIABLE SUBSTITUTION

csh maintains a set of variables, each having as value a list of zero or more words. **csh** sets some of these variables, and merely refers to others. For instance, the *argv* variable is an image of the shell's argument list, and words of its value are referred to in special ways.

You can display and change the values of variables by using the **set** and **unset** commands. Some of the variables the shell refers to are toggles. The shell does not care what their value is, only whether they are set. For instance, the **verbose** variable is a toggle that causes command input to be echoed. Use the command line option –v to set this variable.

Other operations treat variables numerically. The command represented by the at sign, @, permits numeric calculations to be performed, with the result assigned to a variable. However, variable values are always represented as (zero or more) strings. In numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After **csh** has aliased and parsed the input line, and before executing each command, it performs variable substitution keyed by $ characters. You can prevent this expansion by preceding the $ with a \, except within double quotation marks ("), where it always occurs, and within single quotation marks (´) where it never occurs. Strings enclosed in single quotation marks are interpreted later (see COMMAND SUBSTITUTION below), so the dollar sign ($) substitution does not occur until later, if at all. A $ is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable-expanded separately. Otherwise, the command name and entire argument list are expanded together. Therefore, the first (command) word to this point can generate more than one word; the first word becomes the command name, and the rest become arguments.

Unless you enclose the results of variable substitution in double quotation marks or specify the :q modifier, they may eventually be command and filename substituted. Within double quotation marks, a variable whose value consists of multiple words expands to a portion of a single word, with the words of the variable's value separated by blanks. When the :q modifier is applied to a substitution, the variable expands to multiple words with each word separated by a blank and enclosed in quotation marks to prevent later command or filename substitution.

The following metasequences are provided to introduce variable values into the shell input. Except as noted, you cannot reference a variable that is not set. You can apply the **:h**, **:t**, **:r**, **:gh**, **:gt**, and **:gr** modifiers to most of the substitutions below. Substitutions that you cannot do this with are marked accordingly. If braces appear in the command form, you must put the modifiers within the braces. You can apply only one modifier beginning with a colon (:) on each expansion preceded by a dollar sign ($).

$*name*
${*name*}
: Replace text by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters, which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character (_) is considered a letter. If *name* is not a shell variable, but is set in the environment, that value is returned. However, colon (:) modifiers and the other forms given below are not available in this case.

$*name*[*selector*]
${*name*[*selector*]}
: Select only some of the words from the value of *name*. The selector is subjected to $ substitution and may consist of a single number or two numbers separated by a dash (–). The first word of a variable's value is numbered 1. If you omit the first number of a range, the number defaults to 1. If you omit the last member of a range, the number defaults to $#*name*. The selector * selects all words. It is not an error for a range to be empty if you omit the second argument or it is in range.

$#*name*
${#*name*}
: Give the number of words in the variable. This is useful for later use in a "[*selector*]".

$0
: Substitute the filename from which command input is being read. An error occurs if the name is not known.

$*number*
${*number*}
: This sequence is equivalent to $argv[*number*].

$*
: This sequence is equivalent to $argv[*].

$?*name*
$?{*name*}
: Substitute the string 1 if *name* is set; 0 if it is not. This substitution cannot be modified with modifiers preceded by a :.

$?0
: Substitute 1 if the current input filename is known; 0 if it is not. This substitution cannot be modified with modifiers preceded by a :.

$$
: Substitute the decimal process number of the parent shell. This substitution cannot be modified with modifiers preceded by a :.

$< Substitute a line from the standard input, with no further interpretation. This sequence is useful for reading from the keyboard in a shell script. This substitution cannot be modified with modifiers preceded by a :.

COMMAND AND FILENAME SUBSTITUTION

csh applies the remaining substitutions, command and filename substitution, selectively to the arguments of built-in commands. This means that portions of expressions not evaluated are not subjected to these expansions. Names for commands that are not internal to the shell are substituted separately from the argument list. This occurs very late, after input/output redirection is performed, and in a child of the main shell.

COMMAND SUBSTITUTION

Enclosing a command in closing quotation marks (q) indicates command substitution. **csh** usually breaks the output from such a command into separate words at blanks, tabs, and newlines. It discards null words, and uses the modified text to replace the original string. Within double quotation marks, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

FILENAME SUBSTITUTION

If a word contains any of the characters *, ?, [, {, or it begins with ˜, that word is a candidate for filename substitution, also known as "globbing." **csh** regards the word as a pattern, replacing it with an alphabetically sorted list of filenames that match the pattern. In a list of words specifying filename substitution, at least one pattern must match an existing filename, but each pattern need not match. Only the metacharcters *, ?, and [imply pattern matching. The characters ˜ and { are like abbreviations.

In matching filenames, you must match a . at the beginning of a filename or immediately following a / explicitly. This is also true for the / itself. An * matches any string of characters, including the null string. A ? matches any single character. The sequence [...] matches any one of the characters enclosed. Within such a sequence, a pair of characters separated by a – matches any character lexically between the two.

The character ˜ at the beginning of a filename refers to home directories. Standing alone, it expands to your home directory (reflected in the value of the variable **home**). When the ˜ is followed by a name consisting of letters, digits, and –, **csh** searches for a user with that name and substitutes his home directory. Thus, **˜ken** might expand to **/usr/ken** and **˜ken/chmach** to **/usr/ken/chmach.** If the ˜ is followed by a character other than a letter or /, or if it appears somewhere other than at the beginning of a word, the shell leaves it undisturbed.

The metanotation *a{b,c,d}e* is shorthand for *abe ace ade*. Left-to-right order is preserved. The results of matching are sorted separately at a low level to preserve this order (nesting is acceptable). Thus, **˜source/s1/{oldls,ls}.c** expands to

/usr/source/s1/oldls.c /usr/source/s1/ls.c whether or not these files exist, without any chance of error if the home directory for **source** is **/usr/source.** Similarly, **../{memo,*box}** might expand to **../memo ../box ../mbox.** (Note that **memo** was not sorted with the results of matching ***box.**) As a special case, the shell passes all single unmatched braces or an empty pair of braces undisturbed.

INPUT/OUTPUT

To redirect the standard input and standard output of a command, use the following syntax:

< *name* — Open the file *name* (which is first variable-, command-, and filename-expanded) as the standard input.

<< *word* — Read the shell input up to a line identical to *word. word* is not subjected to variable-, filename-, or command-substitution. Each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting \, ", or ´ character appears in *word*, **csh** performs variable and command substitution on the intervening lines, allowing \ to quote a $, a \, and `. Commands that are substituted have all blanks and tabs preserved. All newlines except for the final one are also preserved. The resulting text is placed in an anonymous temporary file, which is given to the command as standard input.

> *name*
>! *name*
>& *name*
>&! *name* — Use the file *name* as standard output. If the file does not exist, create it; if the file does exist, truncate it, discarding its previous contents.

If the variable **noclobber** is set, the file must not exist, or it must be a character special file (for example, a terminal or **/dev/null**), or an error results. This helps prevent accidental destruction of files. The ! forms suppress this check.

Forms involving & route the diagnostic output, as well as the standard output, into the specified file. *name* is expanded in the same way as input filenames beginning with < are.

>> *name*
>>& *name*
>>! *name*
>&!> *name* — Use the file *name* as standard output, but place output at the end of the file. If the variable **noclobber** is set, it is an error for the file not to exist unless you specify one of the forms beginning with !.

A command receives the environment in which the shell was invoked, as modified by the input/output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather, they receive the original standard input of the shell. The << mechanism should be used to present in-line data. This permits shell command scripts to function as components of pipelines and allows the shell to block-read its input. Note that the default standard input for a command run detached is not modified to be the empty file **/dev/null**. Rather, the standard input remains as the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, the process blocks and you are notified (see JOBS above.)

Diagnostic output may be directed through a pipe withthe standard output. Simply use the form |& instead of | to do this.

EXPRESSIONS

A number of the built-in commands take expressions that have operators similar to those used for the C language, with the same precedence. These expressions appear in the @, **exit**, **if**, and **while** commands. The following operators are available:

```
|| && | ^^ & == != =~ !~ <= >= < > << >> + - * / % ! ~ ( )
```

Here the precedence increases to the right, The following characters are, in groups, at the same level:

```
==  !=  =~  !~
<=  >=  <   >
<<  >>
+   -
*   /   %
```

The following operators compare their arguments as strings:

```
== !=  =~  !~
```

All others operate on numbers. The operators =~ and !~ are like == and != except that the right-hand side is a pattern (containing, for example, asterisks, question marks, and instances of [...] characters) against which the left-hand operand is matched. This removes the need to use the **switch** statement in shell scripts when all you need is pattern-matching.

csh considers strings beginning with a zero to be octal numbers. It interprets null or missing arguments as zero. The results of all expressions are strings, which represent decimal numbers. Note that no two components of an expression can appear in the same word. You should surround them by spaces, except when they are adjacent to components of expressions that are syntactically significant to the parser (&, |, <, >, (,)).

Also available in expressions as primitive operands are command executions enclosed in braces ({ and }), and file enquiries of the form *–l name* where *l* is one of the following:

r	Read access
w	Write access
x	Execute access
e	Existence
o	Ownership
z	Zero size
f	Plain file
d	Directory

csh performs command and filename expansion on the specified name, and then checks to see if it has the specified relationship to the real user. If the file does not exist, or if it is inaccessible, all inquiries return false (0). Command executions succeed, returning true (1), if the command exits with status 0; otherwise, they fail, returning false (0). If you require more detailed status information, execute the command outside an expression and examine the **status** variable.

CONTROL FLOW

csh contains a number of commands to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The **foreach**, **switch**, and **while** statements, as well as the **if–then–else** form of the **if** statement require that the major keywords appear in a single simple command on an input line, as shown below.

If the shell's input is not seekable, the shell buffers input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward **goto**s succeed on non-seekable inputs.)

BUILT-IN COMMANDS

Built-in commands are executed within the shell. If a built-in command occurs as any component of a pipeline except the last, it is executed in a sub-shell.

alias Print all aliases.

alias *name*
Print the alias for *name*.

alias *name wordlist*
: Assign the specified *wordlist* as the alias of *name. The wordlist* is command- and filename-substituted. *name* cannot be **alias** or **unalias.**

alloc Show the amount of dynamic memory acquired, broken down into used and free memory. With an argument, this command shows the number of free and used blocks in each size category. The categories start at size eight and double at each step. This command's output may vary across system types.

bg
bg *%job ...*
: Put the current or specified jobs into the background, continuing them if they were stopped.

break Resume execution after the **end** of the nearest enclosing **foreach** or **while.** Execute the remaining commands on the current line. Thus you can have multi-level breaks by writing them all on one line.

breaksw
: Break from a **switch,** resuming after the **endsw.**

case *label:*
: Specify a label in a **switch** statement.

cd
cd *name*
chdir
chdir *name*
: Change the shell's working directory to directory *name.* If you do not specify an argument, change to the home directory of the user.

  If *name* is not found as a subdirectory of the current directory and does not begin with /, ./, or ../, check each component of the variable **cdpath** to see if it has a subdirectory *name.* Finally, if all else fails but *name* is a shell variable whose value begins with /, check to see if it is a directory.

continue
: Continue execution of the nearest enclosing **while** or **foreach.** Execute remaining commands on the current line.

default: Label the default case in a **switch** statement. This command should follow all case labels.

dirs Print the directory stack. The top of the stack is at the left, and the first directory in the stack is the current directory.

echo *wordlist*
echo –n *wordlist*
: Write the specified words to the shell's standard output, separated by spaces, and terminated with a newline, unless you specify the **–n** option.

else
end
endif
endsw See the description of the **foreach**, **if**, **switch**, and **while** statements below.

eval *arg* ...
Read the arguments as input to the shell, executing the resulting command(s) in the context of the current shell. This occurs as in **sh**(1). The command is generally used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See **tset**(1) for an example of using **eval**.

exec *command*
Execute the specified *command* in place of the current shell.

exit Exit with the value of the **status** variable.

exit(*expr*)
Exit with the value of the specified **expr**.

fg
fg *%job* ...
Bring the current or specified jobs into the foreground, continuing them if they were stopped.

foreach *name* (*wordlist*)
...
end Successively set the variable *name* to each member of *wordlist*, and execute the sequence of commands between this command and the matching *end*. (Both **foreach** and **end** must appear alone on separate lines.)

Use the **continue** command to continue the loop prematurely. Use the **break** command to terminate it prematurely. When the shell reads this command from the terminal, it reads the loop once, prompting with "? " before executing any statements in the loop. If you make a mistake typing in a loop at the terminal, you can interrupt it.

glob *wordlist*
Perform the same function as the **echo** command, but do not recognize backslash escapes, and delimit words by null characters in the output. Use this command with programs that use the shell to filename-expand a list of words.

goto *word*
Perform filename- and command-expansion on the specified *word* to yield a string of the form *label*. Cause the shell to rewind input as much as possible and search for a line of the form *label:* (possibly preceded by blanks or tabs). Continue execution after the specified line.

hashstat
Print a statistics line indicating how effective the internal hash table has been at locating commands and avoiding instances of the exec command. An exec is attempted for each component of the path where the hash function indicates a possible hit, and in each component that does not begin with a slash.

history Display the history event list.

history *n*
Print only the *n* most recent events in the history event list.

history −r *n*
Print the most recent events first (rather than printing the oldest first).

history −h *n*
Print the history event list without leading numbers. Use this command to produce files suitable for sourcing with the **−h** option to the **source** built-in command.

if (*expr*) *command*
If the specified expression evaluates true, execute the single *command* with arguments. Variable substitution on *command* happens early, at the same time it does for the rest of the **if** command. The *command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, in which case, *command* is not executed.

if (*expr*) **then**
...
else if (*expr2*) **then**
...
else
...
endif If the specified *expr* is true, execute the commands to the first **else**; otherwise if *expr2* is true, execute the commands to the second **else**, and so on. Any number of **else-if** pairs are possible; only one **endif** is needed. The **else** part is optional. (The words **else** and **endif** must appear at the beginning of input lines; the **if** must appear alone on its input line or after an **else**.)

inlib *lib* Install a user-supplied library specified by *lib* in the shell process. The library is used to resolve external references of programs (and libraries) loaded after its installation. Note that the library is not loaded into the address space unless it is needed to resolve an external reference. The list of **inlibed** libraries is passed to all children of the current shell. Use **llib**(1) to examine this list.

jobs List the active jobs.

jobs −l List the active jobs, and include process IDs.

kill *%job*
kill *–sig %job ...*
kill *pid*
kill *–sig pid ...*
kill –l Send either the TERM (terminate) signal or the specified signal to the jobs or processes indicated. Provide signals by number or by names (as given in **/usr/include/signal.h,** stripped of the SIG prefix). A **kill –l** lists the signal names. There is no default process for this command. If the signal being sent is TERM (terminate) or HUP (hangup), the job or process is sent a CONT (continue) signal as well.

limit
limit *resource*
limit *resource maximum-use*
limit –h
limit –h *resource*
limit –h *resource maximum-use*

Limits the consumption by the current process and each process it creates, to not individually exceed *maximum-use* on the specified *resource.* If you do not specify *maximum-use,* the current limit is printed; if you do not specify *resource,* all limitations are given. If you specify the **–h** flag, the hard limits are used instead of the current limits. The hard limits impose a ceiling on the values of the current limits. Only the super-user can raise the hard limits, but a user can lower or raise the current limits within the legal range.

Resources controllable currently include **cputime** (the maximum number of CPU seconds to be used by each process), **filesize** (the largest single file that you can create), **datasize** (the maximum growth of the data+stack region via **sbrk(2)** beyond the end of the program text), **stacksize** (the maximum size of the automatically-extended stack region), and **coredumpsize** (the size of the largest core dump that will be created). NOTE: You cannot use **limit** to set **stacksize;** the **coredumpsize** limit is always 0 in Domain/OS.

You can specify the *maximum-use* as a (floating point or integer) number followed by a scale factor. For all limits other than *cputime* the default scale is '**k**' or '**kilobytes**' (1024 bytes); you can also us a scale factor of '**m**' or '**megabytes**'. For **cputime** the default scaling is '**seconds**', but you can specify '**m**' for minutes or '**h**' for hours, or a time of the form '*mm:ss*' giving minutes and seconds.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

login Terminate a log-in shell, replacing it with an instance of **/bin/login.** This is one way to log out, and it is included for compatibility with **sh**(1).

logout Terminate a log-in shell. This command is especially useful if **ignoreeof** is set.

nice Set the **nice**(1) scheduling priority for this shell to 4.

nice *+number*
Set the **nice**(1) priority to the given *number.*

nice *command*
Run *command* at **nice**(1) priority 4.

nice *+number command*
Run *command* at positive *number* **nice**(1) priority. The greater the number, the lower CPU priority the process gets. The super-user can specify negative priority by using **nice** *–number* .... The command is always executed in a sub-shell, and the restrictions placed on commands in simple **if** statements apply.

nohup When you specify this command in a shell script, ignore hangups for the remainder of the script.

nohup *command*
Run the specified *command* with hangups ignored. This happens to all processes detached with **&**.

notify
notify *%job ...*
Notify the user asynchronously when the status of the current or specified jobs changes (normally, notification is presented before a prompt). This is automatic if the shell variable **notify** is set.

onintr Restore the default action of the shell on interrupts (to terminate shell scripts or to return to the terminal command input level). In any case, if the shell is running detached and interrupts are being ignored, all forms of **onintr** have no meaning, and interrupts continue to be ignored by the shell and all invoked commands.

onintr –
Ignore all interrupts.

onintr *label*
Execute a **goto** *label* when an interrupt is received or a child process terminates because it was interrupted.

popd Pop the directory stack, returning to the new top directory. The elements of the directory stack are numbered from zero, starting at the top.

popd +*n*
Discard the *n*th entry in the directory stack.

pushd Exchange the top two elements of a directory stack.

pushd *name*
Change to *name* directory and push the old current working directory onto the directory stack.

pushd *+n*
Rotate the *n*th argument of the directory stack around to be the top element and change to it. The members of the directory stack are numbered from zero, starting at the top.

rehash Recompute the internal hash table of the contents of the directories in the **path** variable. This is needed if new commands are added to directories in the **path** while you are logged in. This should be necessary only if you add commands to one of your own directories, or if someone changes the contents of one of the system directories.

repeat *count command*
Execute the specified *command* (subject to the same restrictions as the *command* in the one-line **if** statement above) *count* times. I/O redirections occur exactly once, even if *count* is zero.

rootnode *arg*
Change the current node entry directory to *arg*. See **rootnode**(1).

set Show the value of all shell variables. Variables that have other than a single word as their value appear as a parenthesized word list.

set *name*
Set *name* to the null string.

set *name=word*
Set *name* to the single *word.* In all cases, the value is command- and filename-expanded.

set *name[index]=word*
Set the *index*th component of *name* to *word.* This component must already exist. In all cases, the value is command- and filename-expanded.

set *name=(wordlist)*
Set *name* to the list of words in *wordlist.* In all cases, the value is command- and filename-expanded. You can repeat these arguments to set multiple values in a single **set** command. Note, however, that variable expansion happens for all arguments before any setting occurs.

setenv List all current environment variables.

setenv *name*
Set *name* to an empty string.

setenv *name value*
Set the value of the environment variable *name* to be *value,* a single string. The most commonly used environment variables — USER, TERM, and PATH — are automatically imported to and exported from the **csh** variables **user**, **term**, and **path**. You do not have to use **setenv** for these.

shift Shift the members of **argv** to the left, discarding **argv[1]**. It is an error for the **argv** variable not to be set or to have less than one word as its value.

shift *variable*
Shift the specified *variable* to the left, discarding *variable*[**1**].

source *name*
Read commands from *name*. You may nest **source** commands, but if you nest them too deeply, the shell may run out of file descriptors. An error in a **source** at any level terminates all nested **source** commands.

source –h *name*
Place commands in the history list without executing them. Normally, input during **source** commands is not placed on the history list.

stop Stop the current job that is executing in the background.

stop *%job ...*
Stop the specified *job* that is executing in the background.

suspend
Causes the shell to stop in its tracks, much as if it had been sent a stop signal with CTRL/Z. This is most often used to stop shells started by **su**(1)

```
switch (string)
case str1:
  ...

 breaksw
  ...
default:
  ...

 breaksw
endsw
```

Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters *, ? and [...] may be used in the case labels, which are variable expanded. If none of the labels match before a "default" label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command **breaksw** causes execution to continue after the **endsw**. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the **endsw**.

time Print a summary of time used by this shell and its children.

time *command*
If arguments are given the specified simple command is timed and a time

summary as described under the **time** variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

umask Display the file-creation mask (in octal).

umask *value*
Set the file-creation mask to the specified *value*. Common values for the mask are 002 (giving all access to the group and read and execute access to others) or 022 (giving all except write access to users in the group or others).

unalias *pattern*
Discard all aliases whose names match the specified *pattern*. Thus, **unalias *** removes all aliases. It is not an error for nothing to be **unaliased**.

unhash Disable the internal hash table mechanism, normally used to speed location of executed programs.

unlimit
unlimit *resource*
unlimit –h
unlimit –h *resource*
Removes the limitation on *resource*. If you do not specify *resource*, all *resource* limitations are removed. If you specify **–h**, the corresponding hard limits are removed. Only the super-user can do this.

unset *pattern*
Remove all variables whose names match the specified pattern. Thus, **unset *** removes all variables. This has noticeably distasteful side-effects. It is not an error for nothing to be **unset**.

unsetenv *pattern*
Remove all variables whose names match the specified *pattern* from the environment. Also refer to the **setenv** built-in shell command, above, and the **printenv**(1) command.

ver [*systype* [*command*]]
With no arguments, return the current value of the SYSTYPE environment variable. With a *systype* argument, change the SYSTYPE environment variable to either **bsd4.3** or sys5.3, depending on which is specified.

wait Wait for all background jobs. If the shell is interactive, an interrupt can disrupt the wait, and the shell prints the names and job numbers of all jobs known to be outstanding.

which Identify which file would be executed if the command were submitted for execution. The command is submitted to normal alias and variable substitutions.

while (*expr*)
...
end While the specified expression evaluates to nonzero, evaluate the commands between the **while** and the matching **end.** You can use **break** and **continue** to

terminate or continue the loop prematurely. (The **while** and **end** must appear alone on their input lines.) Prompting occurs the first time through the loop, as for the **foreach** statement if the input is a terminal.

%job Bring the specified *job* number into the foreground.

%job &
: Continue the specified *job* in the background.

@ Print the values of all the shell variables.

@ *name* = *expr*
: Set the specified *name* to the value of *expr*. If the expression contains a >, <, &, or | character, you must enclose at least that part in parentheses.

@ *name*[*index*] = *expr*
: Assign the value of *expr* to the *index*th argument of *name*. Both *name* and its *index*th component must already exist.

The operators *=, +=, etc., are available as in C. The space separating the name from the assignment operator is optional. However, spaces are mandatory in separating components of *expr* that would otherwise be single words. Special postfix ++ and -- operators increment and decrement *name*, respectively, for example, @ **i++**.

PREDEFINED AND ENVIRONMENT VARIABLES

The following variables have special meaning to **csh.** Of these, the shell sets **argv**, **cwd**, **home**, **path**, **prompt**, **shell**, and **status**. Except for **cwd** and **status**, this setting occurs only at initialization. These variables will not then be modified unless you explicitly perform the modification.

csh copies the USER environment variable into the **user** variable; TERM into **term**; and HOME into **home**. It then copies these back into the environment whenever the normal shell variables are reset.

csh handles the PATH environment variable in a similar manner. Do not worry about the setting for PATH other than in the file **.cshrc**. Inferior **csh** processes import the definition of **path** from the environment, and re-export it if you then change it.

argv	Set to the arguments to the shell. It is from this variable that positional parameters are substituted, that is, **$argv[1]** replaces **$1**, etc.
cdpath	Give a list of alternate directories searched to find subdirectories in **chdir** commands.
cwd	Give the full pathname of the current directory.
echo	Echo each command and its arguments just before the command is executed. This variable is set when you specify the **-x** command line option. For non-built-in commands all expansions occur before echoing. Echo built-in commands before command and filename substitution, since these substitutions are then done selectively.

filec	Enable filename completion.
histchars	Change the characters used in history substitution, if you specify a string value. Use the first character of its value as the history substitution character, replacing the default character !. The second character of its value replaces the ^ character in quick substitutions.
history	Control the size of the history list. If you specify a numeric value, do not discard any command that has been referenced in that many events. The last executed command is always saved on the history list. The shell may run out of memory if the value of **history** is too large.
home	Represents the home directory of the invoker, initialized from the environment. The filename expansion of ~ refers to this variable.
ignoreeof	If set, ignore the end-of-file from terminal input devices. This prevents shells from accidentally being killed by an EOF.
mail	Represent the files where the shell checks for mail. This is done after each command completion that results in a prompt, if a specified interval has elapsed. The shell will tell you that you have new mail, if the file exists with an access time not greater than its modify time. If the first word of the value of **mail** is numeric, it specifies a different mail-checking interval (in seconds) than the default (10 minutes). If you specify multiple mail files, the shell tells you that you have new mail in *name*, when there is mail in the file *name*.
noclobber	Restrict output redirection to ensure that files are not accidentally destroyed, and that redirections done with >> refer to existing files.
noglob	If set, inhibit filename expansion. Use this in shell scripts that do not deal with filenames, or after you have obtained a list of filename,s and further expansions are not desirable.
nonomatch	If set, it is not an error for a filename expansion not to match any existing files; rather, the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, that is, **echo** [still gives an error.
notify	If set, notify the user asynchronously of job completions. By default, the shell presents job completions just before printing a prompt.
path	Each word of the **path** variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no **path** variable, **csh** executes only full pathnames. The default search path in Domain/OS SysV is (. **/usr/ucb** **/bin** **/usr/bin** **/usr/apollo/bin**). However, this may vary from system to system. For the super-user, the default search path is (**/etc** **/bin** **/usr/bin** **/usr/apollo/bin**), which may also vary. A shell

that is given neither the –c nor the –t option normally hashes the contents of the directories in the **path** variable after reading **.cshrc**, and each time the **path** variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the **rehash** comand, or the new commands may not be found.

prompt	The string printed on the **csh** command line, before the shell reads commands from an interactive terminal input. If ! appears in the string, replace it by the current event number (unless a preceding backslash is given). The default prompt is "%"; for the super-user, the default prompt is "#".
savehist	Give a numeric value to control the number of history list entries saved in ~/**.history** at log-out time. Save any command that has been referenced in that many events. During start-up, the shell sources ~/**.history** into the history list, enabling history to be saved across log-ins. If the value of **savehist** is too large, the shell is slow during start-up.
shell	Represent the file in which the shell resides. This is used in forking shells to interpret files that have execute bits set, but are not executable by the system. (See the description of Non-Built-in Command Execution, below.) This variable is initialized to the (system-dependent) home of the shell.
status	Give the status returned by the last command. If it terminated abnormally, add 0200 to the status. Built-in commands that fail return exit status 1. All other built-in commands set status 0.
time	Control automatic timing of commands, if a numeric value is supplied. If set, print the user, system, a utilization percentage, and real times for any command that takes more than this many CPU seconds, when the command terminates. A utilization percentage is the ratio of user time plus system time to real time.
verbose	Print the words of each command after history substitution. This variable is set by the –v command-line option to **csh**.

NON-BUILT-IN COMMAND EXECUTION

When a command to be executed is found to be something other than a built-in command, **csh** attempts to execute it via execve(2). Each word in the variable **path** names a directory from which the shell attempts to execute the command. If you do not specify either a –c or a –t option, the shell hashes the names in these directories into an internal table so that it tries an exec in a directory only if the command potentially resides there. This greatly speeds command location when a lot of directories are present in the search path. For each directory component of **path** that does not begin with a /, the shell concatenates with the given command name to form a pathname of a file that it then attempts to execute. The shell also does this if the internal hash table mechanism is turned off (via **unhash**), or a **–c** or **–t** command-line option is specified in **csh**.

Commands in parentheses are always executed in a sub-shell. Thus, **(cd ; pwd) ; pwd** prints the home directory, leaving you where you were (printing this after the home directory). On the other hand, **cd ; pwd** leaves you in the *home* directory. Commands in parentheses are most often used to prevent **chdir** from affecting the current shell.

If a file has execute permissions but is not an executable binary to the system, **csh** assumes it to be a file containing shell commands, and spawns a new shell to read it.

If there is an **alias** for **shell**, the words of the alias are prefixed to the argument list to form the shell command. The first word of the **alias** should be the full pathname of the shell (for example, **$shell**). Note that this is a special, late-occurring, case of **alias** substitution, and it only allows words to be prefixed to the argument list without modification.

COMMAND LINE OPTIONS

–b This flag forces a break from option processing, causing any further shell arguments to be treated as non-option arguments. The remaining arguments are not interpreted as shell options. You can use the **-b** to pass options to a shell script without confusion or possible subterfuge. The shell will not run a set-user ID script without this option.

–c Commands are read from the (single) following argument, which must be present. Any remaining arguments are placed in *argv*.

–D*name*=*value*
Set the parameter *name* to *value*, then pass it to the shell's environment. This option is useful for tailoring the environment of a shell invoked from a program that isn't a shell (such as the DM). For example, if your key definition sets the **–D** variable as follows: **kd cp /bin/csh –DNEWPAD=true ke**, the **.cshrc** script can use the value of the NEWPAD variable to execute additional commands or perform special processing. You can specify a number of **–D** options.

–e The shell exits if any invoked command terminates abnormally or yields a nonzero exit status.

–f The shell starts faster, because it neither searches for nor executes commands from the file **.cshrc** in the invoker's home directory.

–i The shell is interactive and prompts for its top-level input, even if it appears not to be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

–n Parse commands, but do not execute them. This aids in syntactic checking of shell scripts.

–s Take command input from the standard input.

–t Read and execute a single line of input. Use a backslash (\) to escape the newline at the end of this line and continue onto another line.

–v Set the **verbose** variable, causing command input to be echoed after history substitution.

–x Set the **echo** variable, so that commands are echoed immediately before execution.

–V Set the **verbose** variable even before **.cshrc** is executed.

–X Set the **echo** variable even before **.cshrc** is executed.

ARGUMENT LIST PROCESSING

If argument 0 to the shell starts with a dash (–), this is a log-in shell. If arguments remain after command-line options are processed, but you did not specify one of the **–c**, **–i**, –s, or **–t** options, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by $0. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, **csh** executes a standard shell if the first character of a script is not a pound sign (#), that is, if the script does not start with a comment. Remaining arguments initialize the **argv** variable.

SIGNAL HANDLING

csh normally ignores **quit** signals. Jobs running detached, either by &, or the **bg** or **%... &** commands, are immune to signals generated from the keyboard, including hangups. Other signals have the values the shell inherited from its parent. Use **onintr.** to control the shell's handling of interrupts and terminate signals in shell scripts. Log-in shells catch the TERM (terminate) signal. Otherwise, this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a log-in shell is reading the file **.logout.**

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command that involves filename expansion is limited to one-sixth the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of **alias** substitutions on a single line to 20.

BUGS

When a command is restarted from a stop, the shell prints the directory it started in if this is different from the current directory; this can be misleading (that is, wrong) as the job may have changed directories internally.

Shell built-in functions are not stoppable/restartable. Command sequences of the form 'a ; b ; c' are also not handled gracefully when stopping is attempted. If you suspend 'b', the shell immediately executes 'c'. This is especially noticeable if this expansion results from an alias. It suffices to enclose the sequence of commands in parentheses to force it to a subshell, for example, '(a ; b ; c)'.

Control over tty output after processes are started is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; you should use shell procedures rather than aliases.

Commands within loops, prompted for by "?", are not placed in the history list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. All and more than one : modifier should be allowed on $ substitutions.

The way the **filec** facility is implemented is ugly and expensive.

FILES

~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by log-in shell, after **.cshrc** at login.
~/.logout	Read by log-in shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a '#'.
/tmp/sh*	Temporary file for '<<'.
/etc/passwd	Source of home directories for '~name'.

SEE ALSO

sh(1), access(2), execve(2), fork(2), pipe(2), umask(2), wait(2), tty(4), a.out(5), environ(7);
Using Your SysV Environment.

NAME

csplit – context split

SYNOPSIS

csplit [–s] [–k] [–f *prefix*] *file arg1* [... *argn*]

DESCRIPTION

csplit reads *file* and separates it into *n+1* sections, defined by the arguments *arg1*... *argn*. By default the sections are placed in xx00 ... xx*n* (*n* may not be greater than 99). These sections get the following pieces of *file*:

00:	From the start of *file* up to (but not including) the line referenced by *arg1*.
01:	From the line referenced by *arg1* up to the line referenced by *arg2*.
	⋮
n+1:	From the line referenced by *argn* to the end of *file*.

If the *file* argument is a – then standard input is used.

OPTIONS

–s	Suppresses the printing of all character counts. **csplit** normally prints the character counts for each file created.
–k	Leaves previously created files intact. **csplit** normally removes created files if an error occurs.
–f *prefix*	If the –f Names created files *prefix***00** ... *prefixn*. The default is **xx00** ... xx*n*.

file (arg1 ... argn)

/ rexp /	A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional + or – some number of lines (e.g., **/Page/–5**).
% rexp %	This argument is the same as */rexp/*, except that no file is created for the section.
lnno	A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.

{ num }	Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file is split every *lnno* lines (*num* times) from that point. Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. **csplit** does not affect the original file; it is your responsibility to remove it.

EXAMPLES

```
csplit –f cobol file '/procedure division/' /par5./ /par16./
```

This example creates four files, **cobol00** ... **cobol03**. After editing the "split" files, they can be recombined as follows:

```
cat cobol0[0–3] > file
```

Note that this example overwrites the original file.

```
csplit –k file 100 {99}
```

This example would split the file at every 100 lines, up to 10,000 lines. The **–k** option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit –k prog.c '%main(%' '/^}/+1' {20}
```

Assuming that **prog.c** follows the normal C coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in **prog.c**.

DIAGNOSTICS

Self-explanatory except for:

arg – out of range

which means that the given argument did not reference a line between the current position and the end of the file.

SEE ALSO

ed(1), sh(1).
regexp(5) in the *SysV Programmer's Reference*.

NAME

ctrace – C program debugger

SYNOPSIS

ctrace [*options*] [*file*]

DESCRIPTION

ctrace allows you to follow the execution of a C program, statement-by-statement. The effect is similar to executing a shell procedure with the –x *option*. **ctrace** reads the C program in *file* (or from standard input if you do not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of **ctrace** into a temporary file because the **cc**(1) command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the **bfs**(1) or **tail**(1) commands.

OPTIONS

–f *functions*	Traces only these *functions*.
–v *functions*	Traces all but these *functions*.

You may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. String arguments to the **string**(3C) functions and return values from **fgets**(3S), **gets**(3S), and **sprintf**(3S) are printed as strings.

ADDITIONAL VARIABLE OPTIONS

You can request that variables be printed in additional formats, if appropriate, with these *options*:

–o	Octal
–x	Hexadecimal
–u	Unsigned
–e	Floating point

SPECIAL CIRCUMSTANCE OPTIONS

These *options* are used only in special circumstances:

–l *n*	Checks *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.

–s	Suppresses redundant trace output from simple assignment statements and string copy function calls. This *option* can hide a bug caused by use of the = operator in place of the == operator.
–t *n*	Traces *n* variables per statement instead of the default of 10 (the maximum number is 20). The Diagnostics section explains when to use this *option*.
–P	Runs the C preprocessor on the input before tracing it. You can also use the **–D**, **–I**, and **–U** **cpp**(1) *options*. These *options* are used to tailor the run-time trace package when the traced program will run in a non-UNIX System environment:
–b	Uses only basic functions in the trace code, that is, those in **ctype**(3C), **printf**(3S), and **string**(3C). These are usually available even in cross-compilers for microprocessors. In particular, this *option* is needed when the traced program runs under an operating system that does not have **signal**(2), **fflush**(3S), **longjmp**(3C), or **setjmp**(3C).
–p *string*	Changes the trace print function from the default of 'printf('. For example, 'fprintf(stderr,' would send the trace to the standard error output.
–r *f*	Uses file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **–p** *option*).

EXAMPLE

If the file **lc.c** contains this C program:

```
 1 #include <stdio.h>
 2 main()     /* count lines in input */
 3 {
 4    int c, nl;
 5
 6    nl = 0;
 7    while ((c = getchar()) != EOF)
 8          if (c = '\n')
 9                ++nl;
10    printf("%d\n", nl);
11 }
```

and you enter these commands and test data:

```
cc lc.c
a.out
1
(CTRL/d)
```

the program will be compiled and executed. The output of the program will be the number 2, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke **ctrace** with these commands:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

the output will be:

```
2 main()
6     nl = 0;
      /* nl == 0 */
7     while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output will be:

```
      /* c == 49 or '1' */
8            if (c = '\n')
             /* c == 10 or '\n' */
9                  ++nl;
                   /* nl == 1 */
7     while ((c = getchar()) != EOF)
      /* c == 10 or '\n' */
8            if (c = '\n')
             /* c == 10 or '\n' */
9                  ++nl;
                   /* nl == 2 */
7     while ((c = getchar()) != EOF)
```

If you now enter an end of file character (CTRL/d) the final output will be:

```
      /* c == -1 */
10    printf("%d\n", nl);
      /* nl == 2 */2
       return
```

Note that the program output printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by **ctrace** at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator (=) in place of the equality operator (==). You can easily miss this error during code reading.

EXECUTION-TIME TRACE CONTROL

The default operation for **ctrace** is to trace the entire program file, unless you use the –f or –v *options* to trace specific functions. This does not give you statement-by-statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program.

You can do both of these by adding **ctroff**() and **ctron**() function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with *if* statements, and you can even conditionally include this code because **ctrace** defines the **CTRACE** preprocessor variable. For example:

```
#ifdef CTRACE
        if (c == '!' && i > 1000)
                ctron();
#endif
```

You can turn the trace off and on by setting static variable tr_ct_ to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly.

DIAGNOSTICS

This section contains diagnostic messages from both **ctrace** and cc(1), since the traced code often gets some *cc* warning messages. You can get *cc* error messages in some rare cases, all of which can be avoided.

ctrace Diagnostics

warning: some variables are not traced in this statement

Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the –t *option* to increase this number.

warning: statement too long to trace

This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

cannot handle preprocessor code, use –P option

This is usually caused by #ifdef/#endif preprocessor statements in the middle of a C statement, or by a semicolon at the end of a #define preprocessor statement.

'if ... else if sequence too long

Split the sequence by removing an else from the middle.

possible syntax error, try –P option

Use the **−P** *option* to preprocess the **ctrace** input, along with any appropriate **−D**, **−I**, and **−U** preprocessor *options*. If you still get the error message, check the Warnings section below.

cc Diagnostics

warning: illegal combination of pointer and integer
warning: statement not reached
warning: sizeof returns 0
: Ignore these messages.

compiler takes size of function
: See the **ctrace** "possible syntax error" message above.

yacc stack overflow
: See the **ctrace** "'if ... else if' sequence too long" message above.

out of tree space; simplify expression
: Use the **−t** *option* to reduce the number of traced variables per statement from the default of 10. Ignore the "ctrace: too many variables to trace" warnings you will now get.

redeclaration of signal
: Either correct this declaration of *signal*(2), or remove it and #include <signal.h>.

WARNINGS

You will get a **ctrace** syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers. Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

ctrace assumes that BADMAG is a preprocessor macro, and that EOF and NULL are #defined constants. Declaring any of these to be variables, e.g., "int EOF;", will cause a syntax error.

BUGS

ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. **ctrace** may choose to print the address of an aggregate or use the wrong format (e.g., 3.149050e-311 for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

FILES

/usr/lib/ctrace/runtime.c	Run-time trace package

SEE ALSO

signal(2), ctype(3C), fclose(3S), printf(3S), setjmp(3C), string(3C).
bfs(1), tail(1) in the *Using Your SysV Environment*.

NAME

cu – call another UNIX system

SYNOPSIS

cu [**–s**speed] [**–l**line] [**–h**] [**–t**] [**–d**] [**–o** | **–e**] [**–n**] *telno*
cu [**–s** speed] [**–h**] [**–d**] [**–o** | **–e**] **–l** line
cu [**–h**] [**–d**] [**–o** | **–e**] *systemname*

DESCRIPTION

cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible ASCII file transfers.

After making the connection, **cu** runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with a tilde (˜), passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with a tilde (˜), passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with a tilde (˜) have special meanings. Both the *transmit* and the *receive* processes are described in the sections below.

When **cu** is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using a double tilde (˜˜). Executing a tilde command reminds the user of the local system uname. For example, **uname**(1) can be executed on Z, X, and Y as follows:

```
uname
Z
˜[X]!uname
X
˜˜[Y]!uname
Y
```

In general, a tilde (˜) causes the command to be executed on the original machine; a double tilde (˜˜) causes the command to be executed on the next machine in the chain.

The SysV version of **cu** supports the Vadic 212 Autodialer.

OPTIONS

–s*speed* Specifies the transmission speed (300, 1200, 2400, 4800, 9600); The default value is "Any" speed which will depend on the order of the lines in the **/usr/lib/uucp/Devices** file. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

–l*line* Specifies a device name to use as the communication line. Can be used to override searching for the first available line having the right speed. When this option is used without the –s option, the speed of a line is taken from the **/usr/lib/uucp/Devices** file. With the –s option, the

Devices file is searched for the requested speed for the requested line. If possible, the connection is made at the requested speed; otherwise, an error message is printed and the call is not made. The specified device is generally a directly connected asynchronous line (e.g., **/dev/ttyab**) in which case a telephone number (*telno*) is not required. The specified device need not be in the /dev directory. If the specified device is associated with an auto dialer, a telephone number must be provided. Use of this option with *systemname* rather than *telno* will not give the desired result (see *systemname* below).

–h	Emulates local echo, supporting calls to other computer systems that expect terminals to be set to half-duplex mode.
–t	Sets appropriate mapping of carriage-return to carriage-return-line-feed pairs. Used when dialing an ASCII terminal set to auto answer.
–d	Prints diagnostic traces.
–e	Generates even parity for data sent to the remote system.
–o	Generates odd parity for data sent to the remote system.
–n	Prompts you to provide the telephone number to be dialed rather than taking it from the command line (for added security).

SPECIAL ARGUMENTS

telno	When using an automatic dialer, *telno* is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately, for delays of 4 seconds.
systemname	*Systemname* is a **uucp** system name that may be used rather than a phone number. **Cu** obtains an appropriate direct line or phone number from **/usr/lib/uucp/Systems**. **Cu** tries each phone number or direct line for *systemname* in the *Systems* file until a connection is made or all the entries are tried. Note: the *systemname* option should not be used in conjunction with the –l and –s options, as *cu* will connect to the first available line for the system name specified, ignoring the requested line and speed.

TRANSMIT PROCESS

The *transmit* process interprets the following:

~.	Terminate the conversation.
~!	Escape to an interactive shell on the local system.
~!*cmd*...	Run *cmd* on the local system, via the –c option to the **sh**(1) command.
~$*cmd*...	Run *cmd* locally and send its output to the remote system.

~%**cd**	Change the directory on the local system. **NOTE:** A ~!**cd** causes the command to be run by a sub-Shell, which was probably not what was intended.
~%**take** *from* [*to*]	Copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
~%**put** *from* [*to*]	Copy file *from* (on the local system) to file *to* on the remote system. If *to* is omitted, the *from* argument is used in both places. For both ~%**take** and **put** commands, as each block of the file is transferred, consecutive single digits are printed to the terminal.
~~ *line*	Send the line ~ *line* to the remote system.
~%**break**	Transmit a **BREAK** to the remote system (this option can also be specified as ~%**b**).
~%**debug**	Toggle the **-d** debugging option on or off (this option can also be specified as ~%**d**).
~**t**	Print the values of the termio structure variables for the user's terminal (useful for debugging).
~**l**	Print the values of the termio structure variables for the remote communication line (useful for debugging).
~%**nostop**	Toggle between DC3/DC1 input control protocol and no input control. This is useful when the remote system is one which does not respond properly to the DC3 and DC1 characters.

RECEIVE PROCESS

The *receive* process normally copies data from the remote system to its standard output. Internally the program accomplishes this by initiating an output diversion to a file when a line from the remote begins with ~.

Data from the remote is diverted (or appended, if >> is used) to *file* on the local system. The trailing ~> marks the end of the diversion.

Using ~%**put** requires stty(1) and cat(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places. There is an artificial slowing of transmission during the ~%**put** operation, so that loss of data is unlikely.

Using ~%**take** requires echo(1) and cat(1) on the remote system. The stty **tabs** mode should also be set on the remote system, if tabs are to be copied without expansion.

EXAMPLES

To dial a system whose number is 9 201 555 1212 using 1200 baud (where you expect a dialtone after the 9), use the following command:

```
# cu -s1200 9=12015551212
```

If the speed is not specified, "Any" is the default value.

To log in to a system connected by a direct line, type this (where *XX* is a valid TTY number):

```
# cu -l /dev/ttyXX
```

or

```
# cu -l ttyXX
```

To dial a system with the specific line and a specific speed, type this (where *XX* is a valid TTY number):

```
# cu -s1200 -l /dev/ttyXX
```

To dial a system using a specific line associated with an auto dialer, execute the following command (where *XX* is a line number):

```
# cu -l culXX 9=12015551212
```

To use a system name, use this command (where *YYYZZZ* is the name of the system):

```
# cu YYYZZZ
```

BUGS

If you **cu** to a DOMAIN node whose default start-up shell is **/com/sh** (as opposed to **/bin/sh** or **/bin/csh**), you should either: 1) change your command search rules (i.e., do a

```
csr -a /bin /usr/bin ...
```

inside the AEGIS Shell) so that the **cu** *transmit* process can properly locate SysV commands, or 2) have the remote start-up AEGIS Shell invoke a SysV Shell (i.e., */bin/sh*) so that the **cu** *receive* process can properly parse the request (since the tilde character has a special meaning in the AEGIS Shell).

Cu buffers input internally.

The **cu** command does not do any integrity checking on data it transfers. Data fields with special **cu** characters may not be transmitted properly. Depending on the interconnection hardware, it may be necessary to use a **~.** to terminate the conversion even if **stty 0** has been used. Non-printing characters are not dependably transmitted using either the **~%put** or **~%take** commands. **cu** between an IMBR1 and a penril modem will not return a login prompt immediately upon connection. A carriage return will return the prompt.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Devices
/usr/spool/locks/LCK.. (tty-device)

DIAGNOSTICS

Exit code is zero for normal exit, one otherwise.

SEE ALSO

cat (1), echo (1), stty (1), uname (1), uucp (1C).

NAME

cut – cut out selected fields of each line of a file

SYNOPSIS

cut –c*list* [*file*...]

cut –f*list* [**–d***char*] [–s] [*file*...]

DESCRIPTION

Use **cut** to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, i.e., character positions as on a punched card (–c option) or the length can vary from line to line and be marked with a field delimiter character like *tab* (**–f** option). **cut** can be used as a filter; if no files are given, the standard input is used. In addition, a file name of "–" explicitly refers to standard input.

OPTIONS

list	Creates a comma-separated list of integer field numbers (in increasing order), with optional – to indicate ranges [e.g., **1,4,7**; **1–3,8**; **–5,10** (short for **1–5,10**); or 3– (short for third through last field)].
–c*list*	–c (no space) Specifies character positions (e.g., –c**1–72** would pass the first 72 characters of each line).
–f*list*	Lists fields assumed to be separated in the file by a delimiter character (see **–d**); e.g., **–f1,7** copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless –s is specified.
–d*char*	Delimits the field (**–f** option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.
–s	Suppresses lines with no delimiter characters in case of **–f** option. Unless specified, lines with no delimiters will be passed through untouched.

Either the –c or **–f** option must be specified.

Use **grep**(1) to make horizontal "cuts" (by context) through a file, or **paste**(1) to put files together column-wise (i.e., horizontally). To reorder columns in a table, use **cut** and **paste**.

EXAMPLES

Mapping of user IDs to names:

```
cut –d: –f1,5 /etc/passwd
```

Setting *name* to current login name:

```
name=`who am i | cut –f1 –d" "`
```

DIAGNOSTICS

ERROR: line too long
: A line can have no more than 1023 characters or fields, or there is no new-line character.

ERROR: bad list for c/f option
: Missing **–c** or **–f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

ERROR: no fields
: The *list* is empty.

ERROR: no delimeter
: Missing *char* on **–d** option.

ERROR: cannot handle multiple adjacent backspaces
: Adjacent backspaces cannot be processed correctly.

WARNING: cannot open <filename>
: Either *filename* cannot be read or does not exist. If multiple filenames are present, processing continues.

SEE ALSO

grep(1), paste(1).

NAME

cvt_font – convert fonts from pre-SR10 to SR10 format

SYNOPSIS

cvt_font *destination source1* [*source2*]

DESCRIPTION

The **cvt_font** command creates a new font file formatted for SR10. If one *source* name is given, it is converted and placed in the *destination* file. If two source names are given, then the characters in the second source font are concatenated with the characters in the first font, converted, then placed in the *destination* font file.

The source font(s) must be in pre-SR10 format. Since all pre-SR10 fonts have space pre-allocated for 128 characters, the new font can contain up to 256 characters.

If the *destination* font file already exists, or if **cvt_font** fails to find either *source* file, an error is printed, and the command terminates without changing any fonts.

EXAMPLES

The following example takes the **vt100s** font from /sys/**dm/fonts** and formats it for SR10 in the file **vt100s** in the working directory:

```
$ cvt_font vt100s /sys/dm/fonts/vt100s
```

The following example takes the **courier10** and **courier10.a** font files from /sys/**dm/fonts**, concatenates them, and formats them for SR10 in the file **courier10** in the working directory:

```
$ cvt_font courier10 /sys/dm/fonts/courier10 /sys/dm/fonts/courier10.a
```

SEE ALSO

tr_font(1)

NAME

cvtname – convert pathnames between upper and lowercase and preserve colons

SYNOPSIS

cvtname [*options*]

DESCRIPTION

Prior to SR10, the colon (:) was used as an escape character for the purpose of storing mixed-case names. For example, the filename "Readme" was stored as ":readme". Domain/OS programs mapped ":r" and interpreted it as "R". In pre-SR10 Aegis-only environments, colons used in pathnames were treated as literal characters, since Aegis was not case sensitive.

Colon-character constructs in pathnames from pre-SR10 file systems are converted to the appropriate uppercase letter (or special character) automatically when they are copied to SR10 systems. The **cvtname** command allows you to selectively undo that process and thereby restore literal colons to pathnames. **cvtname** also allows you to convert pathnames to all uppercase or all lowercase. The tool operates on entire pathnames. That is, you cannot convert one capital letter in an SR10 pathname back to a "colon-character" sequence without converting them all.

Regardless of the mode specified, **cvtname** queries you before converting each pathname, unless you specify **–nq**, in which case the changes are applied to all objects subordinate to the pathname specified.

OPTIONS

Without options, **cvtname** converts capital letters back to colon-character sequences.

–m *pathname*	Convert capital letters in the names of all objects in *pathname* back to colon-character sequences. If **–li** is also specified, potential changes are listed but no changes are made. If **–nq** is present, the changes are done automatically and all modified names are listed. (The default is **–m** without **–nq**)
–l *pathname*	Convert *pathname* and subordinate object names to all lowercase. If **–li** is also specified, potential changes are listed but no changes are made. If **–nq** is present, the changes are done automatically and all modified names are listed.
–u *pathname*	Convert *pathname* and subordinate object names to all uppercase. If **–li** is also specified, potential changes are listed but no changes are made. If **–nq** is present, the changes are done automatically and all modified names are listed.

EXAMPLES

The following example allows you to convert the capital letters or colon-character constructs in pathnames in the directory **leduc**, querying you before making any changes. Output is shown under the command line. The left-hand column shows unconverted name; right shows converted. Type y to convert, **n** to keep old name.

$ cvtname /ledu
```
/ledu/:C                             /ledu/:::c  n
/ledu/CAT                            /ledu/:c:a:t  y
/ledu/CAT converted to               /ledu/:c:a:t
```

The following example allows you to selectively convert pathnames in **ledu** to uppercase.

$ cvtname –upper /ledu
```
/ledu                              /LEDU  n
/ledu/:c                           /ledu/:C  n
/ledu/:c:a:t                       /ledu/:C:A:T  n
/ledu/acl_from_whoville            /ledu/ACL_FROM_WHOVILLE  y
/ledu/acl_from_whoville converted to /ledu/ACL_FROM_WHOVILLE
/ledu/backup.pas                   /ledu/BACKUP.PAS  n
/ledu/ff1                          /ledu/FF1  y
/ledu/ff1 converted to             /ledu/FF1
/ledu/TD/backup_history            /ledu/TD/BACKUP_HISTORY  n
```

NAME

cvtrgy – convert registry between SR9.x and SR10 formats

SYNOPSIS

cvtrgy [**–from9to10** | **–from10to9** [**–favor_etc**]] **–readonly** |
-owner *pgo* | **–first** | **–nq** | **–from** *source_rgy* **–to** *dest_rgy*

DESCRIPTION

The **cvtrgy** command allows the system administrator to generate an SR10 format registry database from SR9.7 registry files, or generates SR9.7 registry files with data from the SR10 registry. The tool operates on SR9.7 nodes only. Both the **rgyd** and **llbd** servers must be running on the SR10 node, except when the **–first** option is used. Run **cvtrgy** the first time when you add SR10 nodes to your network, and periodically thereafter to keep the pre-SR10 and SR10 registry information synchronized.

You must specify either **–from9to10** or **–from10to9**. By default, **cvtrgy** creates a read-only registry of the destination type. That is, **cvtrgy –from9to10** creates a read-only SR10 format master registry, while **cvtrgy –from10to9** creates a read-only SR9.x format master registry. You can then propagate the information to replica registries in the appropriate way.

Whenever the conversion from SR10 to SR9 occurs, if the registry files exist at the destination node specified in the command line, the tool quits without updating. This means that before running **cvtrgy –from10to9**, you should rename (or move) the SR9.x registry database on the destination node.

The **cvtrgy** tool assigns UNIX identifiers automatically during the conversion process if you prefer. However, if your pre-SR10 node runs Domain/OS, you should preserve the identifiers associated with accounts in your current (pre-SR10) **/etc/passwd** and **/etc/group** files. In normal operation, **cvtrgy** looks for the **/etc/passwd** and **/etc/group** files and assigns identifiers from them, if they exist. Therefore, you should run **cvtrgy** on a 9.7 node that either contains your master **/etc/passwd** and **/etc/group** files or has a link to them.

If **cvtrgy** doesn't find the **/etc/passwd** and **/etc/group** files and an /etc directory exists, it queries you before assigning new UNIX identifiers, unless the **–nq** (no query) flag is turned on, in which case **cvtrgy** exits with an error.

In order to add or change accounts and other registry data, you must edit the writable registry with the tool appropriate to the registry's format (i.e., with **edrgy** on SR10, **edacct** and **edppo** on SR9.x) on a node running the same software release as the format of the writable registry. Thus, if your SR9.x registries were writable, you'd have to edit them using **edacct** and **edppo**, from a node running SR9.7. Once your SR10 registry is the writable one, use **edrgy**.

The **cvtrgy** tool resides in the **/install/tools/cvtrgy** after an SR10 installation and must be copied to an SR9.7 node before you run it. After running **cvtrgy**, you must also run the **crpasswd** command on an SR9.x node to update the **/etc/passwd** and **/etc/group** files. The SR10 directory **/install/tools** contains a new version of **crpasswd** which you should copy to all SR9.7 nodes that need to run **crpasswd**. (You can rename or replace the old version of **crpasswd**.) See the SR10 Transition Guide for further details on running **cvtrgy**.

OPTIONS

–from9to10	Convert SR9.x registry files to SR10 registry format
–from10to9	Convert SR10 registry data to SR9.7 format and place in SR9.7 registry files
–from *source_rgy*	Specify source for registry data to be converted. For **–from9to10**, must be in the form **//node_name/registry/rgy_site**. For **–from10to9**, must be //node_name. Either or both registry sites may be remote from the node running **cvtrgy**.
–to *dest_rgy*	Specify destination for converted registry data. For **–from9to10**, must be in the form **//node_name/registry/rgy_site**. For **–from10to9**, must be **//node_name**. Either or both registry sites may be remote from the node running **cvtrgy**.
–owner *pgo*	Specify SR10 registry owner, in the SID form p.g.o, where all pgo names and the pgo account already exist in the SR9.7 registry. *pgo* is a string of the form *pers.group.org*. You must specify with every invocation of **–from9to10**. This option is meaningful only with the **–from9to10** option.
–first	Specify that this is the first invocation of **cvtrgy**. In this case only, **cvtrgy** runs without **rgyd** and **llbd** servers running. Use only once. Only meaningful with **–from9to10**.
–readonly	Make SR9.7 registries read-only, permanently. Only meaningful with **–from9to10**. Can only be run in this mode once; after running, cannot use **–from9to10** again.
–nq	No query. Silent mode. Don't query before assigning new UNIX identifiers (**cvtrgy** quits). Don't query for owner (**cvtrgy** quits).
–favor_etc	If you've edited UNIX IDs (numbers) in the SR9.7 /etc/**passwd** or /etc/**group** after you've already run **cvtrgy** at least once, you should propagate the new numbers to the SR10 registry. Running **cvtrgy** with this option, in the **–from9to10** direction, propagates the new UNIX IDs to the SR10 registry. After running **cvtrgy** with this option, you must also run /etc/**syncids** on all SR10 disks. Only meaningful with **–from9to10**.

CONVERTING FROM SR9.7 TO SR10

You must be root to run **cvtrgy**. Use the following command line. The **node_name1** is the SR9.7 node.

```
$ cvtrgy -from9to10 -from //node_name1/registry/rgy_site
      -to //node_name2 -owner pgo -first
```

CONVERTING FROM SR10 TO SR9.7

The person who runs the tool must be logged in as root or locksmith. Use the following command line. The **node_name1** is the SR10 node.

```
$ cvtrgy -from10to9 -from //node_name1 -to //node_name2/registry/rgy_site
```

EXAMPLE

The following is a sample transcript from a **cvtrgy** session that converts SR9.x registry data files to an SR10 format registry database. This is the first time **cvtrgy** has been run on the network. A single collision is shown to illustrate **cvtrgy**'s warning message format; you may see more warnings at your site.

```
$ cvtrgy -from9to10 -from //dog/registry/rgy_site1 -to //cat -first -owner %.sys_admin.%

Phase 1 - opening registry files:

Phase 2 - modifying SR9 registry files:

Converted person file saved in registry
//dog/registry/rgy_site1

Converted project file saved in registry
//dog/registry/rgy_site1

Converted org file saved in registry
//dog/registry/rgy_site1

Phase 3 - converting person file:

?(cvtrgy) Warning - unix id collision:
person bin_sr9 reassigned from 3 to 10002
Converted person file saved in registry
//dog/registry/rgy_site1
```

```
Phase 4 - converting project file:

?(cvtrgy) Warning - unix id collision:
project backup reassigned from 1001 to 3
Converted project file saved in registry
//dog/registry/rgy_site1

Phase 5 - converting org file:

Converted org file saved in registry
//dog/registry/rgy_site1

Phase 6 - converting accounts:

Phase 7 - adding default accounts:

Converted account file saved in registry
//dog/registry/rgy_site1

Phase 8 - closing the sr9 registry files:

Phase 9 - writing conversions to sr10 registry:

Conversion completed successfully:
```

SEE ALSO

passwd(4), group(4)

NAME

cxref – generate C program cross-reference

SYNOPSIS

cxref [options] *files*

DESCRIPTION

cxref analyzes a collection of C files and attempts to build a cross-reference table. **cxref** uses a special version of **cpp** to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or, with the –c option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

In addition to the **–D**, **–I** and **–U** options [which are interpreted just as they are by cc(1) and **cpp**(1)], the following options are interpreted by **cxref**.

OPTIONS

–c	Prints a combined cross-reference of all input files.
–w<*num*>	Formats output no wider than <num> (decimal) columns. Defaults to 80 if <num> is not specified or is less than 51.
–o file	Directs output to *file*.
–s	Operate silently; do not print input file names.
–t	Lists format for 80-column width.

FILES

LLIBDIR	usually /usr/**lib**
LLIBDIR/xcpp	special version of the C preprocessor.

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files.

BUGS

cxref considers a formal argument in a **#define** macro definition to be a declaration of that symbol. For example, a program that **#includes ctype.h**, contains many declarations of the variable c.

SEE ALSO

cc(1), cpp(1).

NAME

date – print and set the date

SYNOPSIS

date [mmddhhmm[yy]] | +*format*]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

```
date

10080045
```

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. **date** takes care of the conversion to and from local standard and daylight time. Only the superuser can change the date.

If an argument begins with +, the output of **date** is under the control of the user. All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and is replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	Insert a new-line character
t	Insert a tab character
m	Month of year – 01 to 12
d	Day of month – 01 to 31
y	Last 2 digits of year – 00 to 99
D	Date as mm/dd/yy
H	Hour – 00 to 23
M	Minute – 00 to 59
S	Second – 00 to 59
T	Time as HH:MM:SS
j	Day of year – 001 to 366
w	Day of week – Sunday = 0
a	Abbreviated weekday – Sun to Sat
h	Abbreviated month – Jan to Dec
r	Time in AM/PM Notation

EXAMPLE

The following input:

date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'

would have generated as output:

```
DATE: 08/01/76
TIME: 14:45:05
```

DIAGNOSTICS

no permission You are not the super-user and you try to change the date.

bad conversion
The date set is syntactically incorrect.

bad format character
The field descriptor is not recognizable.

NAME

dbacl – Domain/Dialog™-based access control list editor

SYNOPSIS

dbacl [*file*]

DESCRIPTION

dbacl provides an interactive menu-based editor for manipulating Access Control Lists (ACLs). It is primarily designed with novice or occasional ACL users in mind. **chacl**(1), **cpacl**(1), and **lsacl**(1) are better suited to complicated actions on large numbers of ACLs.

To use **dbacl**, press the left mouse key (or the F1 key) to select items on the screen such as buttons or ACL entries. The bar at the top of the screen contains a "Quit" button, and names of menus. By pressing the mouse key over one of the menu names, you are presented with a pop-up menu with commands. Pull down and release the key over the name of a command to select it.

If a command appears in grayed-out text, it is not currently selectable. For example, before selecting "Cut" from the "Entry" menu, you must select an entry to cut, by clicking on it with the mouse.

You can use the right mouse key (or the F3 key) as a short cut for selecting an entry and choosing the "Change Entry" command.

If a button has a double outline, you can select it by pressing the RETURN key anywhere in its window. Likewise, the ESC key nearly always cancels the current command.

SEE ALSO

chacl(1), cpacl(1), lsacl(1), acl(5), salacl(1M)

NAME

dbx – debugger

SYNOPSIS

dbx [**–r**] [**–i**] [**–I** *dir*] [**–no_src**] [**–no_frame**] [**–c** *file*] [*objfile*]

DESCRIPTION

dbx is a tool for source level debugging and execution of programs under SysV. The *objfile* is an object file produced by a compiler with the appropriate flag (usually –g) specified to produce symbol information in the object file. The machine level facilities of **dbx** can be used on any program.

The object file contains a symbol table that includes the name of the all the source files translated by the compiler to create it. These files are available for perusal while using the debugger.

If the file **.dbxinit** exists in the current directory then the debugger commands in it are executed. ,B dbx also checks for a **.dbxinit** in your home directory if there isn't one in the current directory.

dbx creates a separate transcript pad for debugger interactions unless the **–no_frame** option is specified. **dbx** also creates a window to display source code unless **–no_src** is specified.

OPTIONS

–r	Executes *objfile* immediately. If it terminates successfully **dbx** exits. Otherwise the reason for termination will be reported and the user offered the option of entering the debugger or letting the program fault. **dbx** reads from **/dev/tty** when **–r** is specified and standard input is not a terminal. Unless **–r** is specified, **dbx** just prompts and waits for a command.
–i	Forces **dbx** to act as though standard input is a terminal.
–I *dir*	Adds *dir* to the list of directories that are searched when looking for a source file. Normally **dbx** looks for source files in the current directory and in the directory where *objfile* is located. The directory search path can also be set with the **use** command.
–c *file*	Executes the **dbx** commands in the *file* before reading from standard input.
–no_src	Disables source display.
–no_frame	Does not create a separate debugger transcript pad.

Execution and Tracing Commands

run [*args*] [<*filename*] [>*filename*]
rerun [*args*] [<*filename*] [>*filename*]

Start executing *objfile*, passing *args* as command line arguments; < or > can be used to redirect input or output in the usual manner. When **rerun** is used without any arguments the previous argument list is passed to the program; otherwise it is identical to **run**. If *objfile* has been written since the last time the symbolic information was read in, **dbx** will read in the new information.

trace [**in** *procedure/function*] [**if** *condition*]
trace *source-line-number* [**if** *condition*]
trace *procedure/function* [**in** *procedure/function*] [**if** *condition*]
trace *expression* **at** *source-line-number* [**if** *condition*]
trace *variable* [**in** *procedure/function*] [**if** *condition*]

Have tracing information printed when the program is executed. A number is associated with the command that is used to turn the tracing off (see the **delete** command).

The first argument describes what is to be traced. If it is a *source-line-number*, then the line is printed immediately prior to being executed. Source line numbers in a file other than the current one must be preceded by the name of the file in quotes and a colon, e.g. "mumble.p":17.

If the argument is a procedure or function name then every time it is called, information is printed telling what routine called it, from what source line it was called, and what parameters were passed to it. In addition, its return is noted, and if it's a function then the value it is returning is also printed.

If the argument is an *expression* with an **at** clause then the value of the expression is printed whenever the identified source line is reached.

If the argument is a variable then the name and value of the variable is printed whenever it changes. Execution is substantially slower during this form of tracing.

If no argument is specified then all source lines are printed before they are executed. Execution is substantially slower during this form of tracing.

The clause "**in** *procedure/function*" restricts tracing information to be printed only while executing inside the given procedure or function.

Condition is a boolean expression and is evaluated prior to printing the tracing information; if it is false then the information is not printed.

stop if *condition*
stop at *source-line-number* [**if** *condition*]
stop in *procedure/function* [**if** *condition*]
stop *variable* [**if** *condition*]

Stop execution when the given line is reached, procedure or function called, variable changed, or condition true.

status [> *filename*]

Print out the currently active **trace** and **stop** commands.

delete *command-number ...*

The traces or stops corresponding to the given numbers are removed. The numbers associated with traces and stops are printed by the **status** command.

catch *number*
catch *signal-name*
ignore *number*
ignore *signal-name*

Start or stop trapping a signal before it is sent to the program. This is useful when a program being debugged handles signals such as interrupts. A signal may be specified by number or by a name (e.g., SIGINT). Signal names are case insensitive and the "SIG" prefix is optional. By default all signals are trapped except SIGCONT, SIGCHILD, SIGALRM and SIGKILL.

cont *integer*
cont *signal-name*

Continue execution from where it stopped. If a signal is specified, the process continues as though it received the signal. Otherwise, the process is continued as though it had not been stopped. Execution cannot be continued if the process has "finished", that is, called the standard procedure "exit".

step Execute one source line.

next Execute up to the next source line. The difference between this and **step** is that if the line contains a call to a procedure or function the **step** command will stop at the beginning of that block, while the **next** command will not.

return [*procedure*]

Continue until a return to *procedure* is executed, or until the current procedure returns if none is specified.

call *procedure(parameters)*

Execute the object code associated with the named procedure or function.

Printing Variables and Expressions

Names are resolved first using the static scope of the current function, then using the dynamic scope if the name is not defined in the static scope. If static and dynamic searches do not yield a result, an arbitrary symbol is chosen and the message "[using *qualified name*]" is printed. The name resolution procedure may be overridden by qualifying an identifier with a block name, e.g., "*module.variable*". For C, source files are treated as modules named by the file name without ".c".

Expressions are specified with an approximately common subset of C and Pascal (or equivalently Modula-2) syntax. Indirection can be denoted using either a prefix "*" or a postfix "^" and array expressions are subscripted by brackets ("[ ]"). The field reference operator (".") can be used with pointers as well as records, making the C operator "->" unnecessary (although it is supported).

Types of expressions are checked; the type of an expression may be overridden by using "*type-name*(*expression*)". When there is no corresponding named type the special constructs "&*type-name*" and "$$*tag-name*" can be used to represent a pointer to a named type or C structure tag.

assign *variable* = *expression*
: Assign the value of the expression to the variable.

dump [*procedure*] [>*filename*]
: Print the names and values of variables in the given procedure, or the current one if none is specified. If the procedure given is ".", then the all active variables are dumped.

print *expression* [, *expression* ...]
: Print out the values of the expressions.

whatis *name*
: Print the declaration of the given name, which may be qualified with block names as above.

which *identifier*
: Print the full qualification of the given identifer, i.e. the outer blocks that the identifier is associated with.

up [*count*]
down [*count*]
: Move the current function, which is used for resolving names, up or down the stack *count* levels. The default *count* is 1.

where Print out a list of the active procedures and function.

whereis *identifier*
: Print the full qualification of all the symbols whose name matches the given identifier. The order in which the symbols are printed is not meaningful.

Accessing Source Files

/regular expression[/]
?regular expression[?]
    Search forward or backward in the current source file for the given pattern.

edit [*filename*]
edit *procedure/function-name*
    Invoke an editor on *filename* or the current source file if none is specified. If a *procedure* or *function* name is specified, the editor is invoked on the file that contains it. Which editor is invoked by default depends on the installation. The default can be overridden by setting the environment variable EDITOR to the name of the desired editor.

file [*filename*]
    Change the current source file name to *filename*. If none is specified then the current source file name is printed.

func [*procedure/function*]
    Change the current function. If none is specified then print the current function. Changing the current function implicitly changes the current source file to the one that contains the function; it also changes the current scope used for name resolution.

list [*source-line-number* [, *source-line-number*]]
list *procedure/function*
    List the lines in the current source file from the first line number to the second inclusive. If no lines are specified, the next 10 lines are listed. If the name of a procedure or function is given lines *n-k* to *n+k* are listed where *n* is the first statement in the procedure or function and *k* is small.

use *directory-list*
    Set the list of directories to be searched when looking for source files. The *directory-list* is used if the specified file cannot be found, or if the file is found but the modified time does not match the time in the object module. If a file is found using *directory-list*, or if the file's modified time is different then the source display banner will display the filename being displayed as well as the stored filename in parentheses.

Command Aliases and Variables

alias *name name*
alias *name "string"*
alias *name* (*parameters*) *"string"*

When commands are processed, **dbx** first checks to see if the word is an alias for either a command or a string. If it is an alias, then **dbx** treats the input as though the corresponding string (with values substituted for any parameters) had been entered. For example, to define an alias "rr" for the command "rerun", one can say

```
alias rr rerun
```

To define an alias called "b" that sets a stop at a particular line one can say

```
alias b(x) "stop at x"
```

Subsequently, the command "b(12)" will expand to "stop at 12".

set *name* [= *expression*]

The set command defines values for debugger variables. The names of these variables cannot conflict with names in the program being debugged, and are expanded to the corresponding expression within other commands. The following variables have a special meaning:

$hexchars
$hexints
$hexoffsets
$hexstrings

When set, **dbx** prints out out characters, integers, offsets from registers, or character pointers respectively in hexadecimal.

$listwindow

The value of this variable specifies the number of lines to list around a function or when the **list** command is given without any parameters. This value is also used when displaying source in the source window. The current line is positioned so that as much of the listwindow as possible is visible. Its default value is 10.

$unsafecall
$unsafeassign

When "$unsafecall" is set, strict type checking is turned off for arguments to subroutine or function calls (*e.g.* in the **call** statement). When "$unsafeassign" is set, strict type checking between the two sides of an **assign** statement is turned off. These variables should be used only with great care, because they severely limit **dbx**'s usefulness for detecting errors.

unalias *name*
: Remove the alias with the given name.

unset *name*
: Delete the debugger variable associated with *name*.

Machine Level Commands

tracei [*address*] [**if** *cond*]
tracei [*variable*] [**at** *address*] [**if** *cond*]
stopi [**if** *cond*]
stop at address [**if** *cond*]
: Turn on tracing or set a stop using a machine instruction address.

stepi

nexti Single step as in **step** or **next**, but do a single instruction rather than source line.

address ,*address*/ [*mode*]
address / [*count*] [*mode*]
: Print the contents of memory starting at the first *address* and continuing up to the second *address* or until *count* items are printed. If the address is ".", the address following the one printed most recently is used. The *mode* specifies how memory is to be printed; if it is omitted the previous mode specified is used. The initial mode is "X". The following modes are supported:

i	Print the machine instruction
d	Print a short word in decimal
D	Print a long word in decimal
o	Print a short word in octal
O	Print a long word in octal
x	Print a short word in hexadecimal
X	Print a long word in hexadecimal
b	Print a byte in octal
c	Print a byte as a character
s	Print a string of characters terminated by a null byte
f	Print a single precision real number
g	Print a double precision real number

Symbolic addresses are specified by preceding the name with an "&". Registers are denoted by $D0-$D7, for the data registers, and $A0-$A7, for the address registers. For convenience, $DB, $SB, $SP, and $PC are also available. Addresses may be expressions made up of other addresses and the operators "+", "-", and indirection (unary "*").

Miscellaneous Commands

help Print out a synopsis of **dbx** commands.

quit Exit **dbx**.

sh *command-line*
Pass the command line to the shell for execution. The SHELL environment variable determines which shell is used.

source *filename*
Read **dbx** commands from the given *filename*.

NOTES

Assignments to structures with bit fields does not work, and assigning through a pointer variable may cause **dbx** to have a stack underflow and abort.

Some problems remain with the support for individual languages. Fortran problems include: inability to assign to logical, logical*2, complex and double complex variables; inability to represent parameter constants which are not type integer or real; peculiar representation for the values of dummy procedures (the value shown for a dummy procedure is actually the first few bytes of the procedure text; to find the location of the procedure, use "&" to take the address of the variable).

FILES

a.out	Object file
.dbxinit	Initial commands

SEE ALSO

cc(1)

NAME

dc – desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. (See **bc**(1), a preprocessor for **dc** that provides infix notation and a C-like syntax that implements functions. **bc** also provides reasonable control structures for programs.) The overall structure of **dc** is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input.

OPTIONS

number	The value of the number is pushed on the stack. A number is an unbroken string of the digits 0–9. It may be preceded by an underscore (_) to input a negative number. Numbers may contain decimal points.
+ – / * % ^	The top two values on the stack are added (+), subtracted (–), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.
s*x*	The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the s is capitalized, *x* is treated as a stack and the value is pushed on it.
l*x*	The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the **l** is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.
d	The top value on the stack is duplicated.
p	The top value on the stack is printed. The top value remains unchanged.
P	Interprets the top of the stack as an ASCII string, removes it, and prints it.
f	All values on the stack are printed.
q	Exits the program. If executing a string, the recursion level is popped by two.
Q	Exits the program. The top value on the stack is popped and the string execution level is popped by that value.
x	Treats the top element of the stack as a character string and executes it as a string of **dc** commands.
X	Replaces the number on the top of the stack with its scale factor.

[...]	Puts the bracketed ASCII string onto the top of the stack.
<*x* >*x* =*x*	The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.
v	Replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.
!	Interprets the rest of the line as a UNIX system command.
c	All values on the stack are popped.
i	The top value on the stack is popped and used as the number radix for further input. **I** Pushes the input base on the top of the stack.
o	The top value on the stack is popped and used as the number radix for further output.
O	Pushes the output base on the top of the stack.
k	The top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
z	The stack level is pushed onto the stack.
Z	Replaces the number on the top of the stack with its length.
?	A line of input is taken from the input source (usually the terminal) and executed.
; :	are used by **bc**(1) for array operations.

EXAMPLE

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

DIAGNOSTICS

x is unimplemented
: *x* is an octal number.

stack empty
: Not enough elements on the stack to do what was asked.

Out of space
: The free list is exhausted (too many digits).

Out of headers
: Too many numbers being kept around.

Out of pushdown
Too many items on the stack.

Nesting Depth
Too many levels of nested execution.

SEE ALSO

bc(1).

NAME

dd – convert and copy a file

SYNOPSIS

dd [*option=value*] ...

DESCRIPTION

dd copies the specified input file to the specified output with possible conversions. By default, it uses the standard input and output. You may specify the input and output block size. After completion, **dd** reports the number of whole and partial input and output blocks.

OPTIONS/VALUE PAIRS

ibs=*n*	Inputs block size *n* bytes; 512 is the default.
obs=*n*	Outputs block size; 512 is the default.
bs=*n*	Sets both input and output block size, superseding **ibs** and **obs**.
cbs=*n*	Conversion buffer size; used only if **conv=ascii** or **conv=ebcdic** is specified. In the former case, **cbs** characters are placed into the conversion buffer, converted to ASCII, and trimmed of any trailing blanks. Newlines are then added before sending the line to the output. In the latter case, ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks are added to make up an output block of size **cbs**.
skip=*n*	Skips *n* input blocks before starting copy.
seek=*n*	Seeks *n* blocks from the beginning of the output file before copying.
count=*n*	Copies only *n* input blocks.
conv=ascii	Converts EBCDIC to ASCII.
ebcdic	Converts ASCII to EBCDIC.
ibm	Maps ASCII to EBCDIC in a slightly different way than the above case.
lcase	Maps alphabetics to lowercase.
ucase	Maps alphabetics to uppercase.
swab	Swaps every pair of bytes.
noerror	Does not stop processing on an error.
sync	Pads every input block to **ibs**.
...,...	Represents several comma-separated conversions.

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by x to indicate a product.

The ASCII/EBCDIC conversion tables are taken from the 256-character standard of the CACM (November, 1968). The **ibm** conversion, while less accepted as a standard, corresponds better to certain IBM print train conventions.

EXAMPLE

To read an EBCDIC tape blocked with ten 80-byte EBCDIC card images per block into the ASCII file *x*, use the following:

```
# dd if=/dev/rmt0 of=x ibs=800 cbs=80
```

BUGS

SysV does not support some raw I/O devices typically used with **dd**.

Newlines are inserted only on conversion to ASCII. Padding is done only on conversion to EBCDIC. These should be separate options.

DIAGNOSTICS

f+p blocks in(out)	Numbers of full and partial blocks read(written).

SEE ALSO

cp (1).

NAME

dde – Domain Distributed Debugging Environment

SYNOPSIS

dde [**–do** *"cmd_list"*]
  [[**–on** *target_machine*] [**–target_type** *target_type*]
  { [**–input** *pathname*] [**–output** *pathname* [–ao]]
   [**–errors** *pathname* [–ae]] *program_invocation*
  | **–attach** *process_id* }]

DESCRIPTION

The **dde** command invokes the Domain Distributed Debugging Environment, the standard debugger for the Domain/OS operating system at SR10. For complete information about this debugger and its commands, consult the *Domain Distributed Debugging Environment Reference* (011024) or invoke the debugger's own **help** command for online assistance.

OPTIONS

–do *"cmd_list"* Execute *cmd_list* (a list of debugger commands) before executing any startup files or debugging the program. The sample option specification -do "property layout -notarget" illustrates a common use of this option (to inhibit the creation of a separate window for the target program).

–on *target_machine* Debug the program or process on the specified target machine, where *target_machine* is a node name or node ID.

–target_type *target_type*
Specify the type of target machine; *target_type* must be "m68k" for SR10.

–input *pathname* Read target program input from *pathname*.

–output *pathname* [–ao]
Direct target program output to *pathname*. With -ao, append output to *pathname*.

–errors *pathname* [–ae]
Direct target program error output to *pathname*. With -ae, append error output to *pathname*. To redirect error output and standard output to the same file, use the same pathname on both options or use "&1" as an argument to the -errors option.

program_invocation Invoke *program_invocation* (the pathname of an executable image, plus any arguments) for debugging. This specification must be last on the dde command line.

−attach *process_id*	Attach to a running process identified by the UNIX pid *process_id*. Use the **/bin/ps** or **/com/pst -un** commands to get the pid of a process.

NAME

delta – make a delta (change) to an SCCS file

SYNOPSIS

delta [**–rSID**] [**–s**] [**–n**] [**–g***list*] [**–m**[*mrlist*]] [**–y**[*comment*]] [**–p**] *files*

DESCRIPTION

delta permanently introduces into the named SCCS file changes that were made to the file retrieved by **get**(1) (called the **g–file**, or generated file).

delta makes a delta to each named SCCS file. If a directory is named, **delta** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read (see WARNINGS); each line of the standard input is taken to be the name of an SCCS file to be processed.

delta can issue prompts on the standard output depending on certain *options* and *flags* [see **admin**(1)] that may be present in the SCCS file (see **–m** and **–y** *options* below).

OPTIONS

Option arguments apply independently to each named file.

–r*SID* Uniquely identifies which delta is to be made to the SCCS file. It is only necessary to use this *option* if two or more outstanding **get**s for editing (**get –e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **–r** option can be either the SID specified on the **get** command line or the SID to be made as reported by the **get** command [see **get**(1)]. A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

–s Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

–n Specifies retention of the edited **g–file** (normally removed at completion of delta processing).

–g*list* Creates a *list* (see **get**(1) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (SID).

–m*[mrlist]* If the SCCS file has the **v** flag set [see **admin**(1)] then a Modification Request (**MR**) number *must* be supplied as the reason for creating the new delta.

If **–m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **–y** option).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MR** list.

Note that if the v flag has a value [see **admin**(1)], it is taken to be the name of a program (or shell procedure) which will validate the correctness of the **MR** numbers. If a non-zero exit status is returned from the **MR** number validation program, **delta** terminates. (It is assumed that the **MR** numbers were not all valid.)

−y*[comment]* Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

If −y is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

−p Causes **delta** to print (on the standard output) the SCCS file differences before and after the delta is applied in a **diff**(1) format.

BUGS

Lines beginning with an **SOH** ASCII character (binary 001) cannot be placed in the SCCS file unless the **SOH** is escaped. This character has special meaning to SCCS [see **sccsfile**(4) (5)] and will cause an error.

Avoid using a **get** of many SCCS files, followed by a **delta** of those files, when the **get** generates a large amount of data. Instead, use multiple **get**/**delta** sequences.

If the standard input (−) is specified on the **delta** command line, **−m** (if necessary) and −y *must* also be present. Omission of these *options* causes an error.

Comments are limited to text strings of at most 512 characters.

FILES

g−file	Existed before the execution of **delta**; removed after completion of **delta**.
p−file	Existed before the execution of **delta**; may exist after completion of **delta**.
q−file	Created during the execution of **delta**; removed after completion of **delta**.
x−file	Created during the execution of **delta**; renamed to SCCS file after completion of **delta**.
z−file	Created during the execution of **delta**; removed during execution of **delta**.
d−file	Created during the execution of **delta**; removed after completion of delta.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the **g−file**.

DIAGNOSTICS

Use **help**(1) for explanations.

SEE ALSO

admin(1), cdc(1), get(1), prs(1), rmdel(1), sccs(1), sccsfile(4).
bdiff(1), help(1) in *Using Your SysV Environment.*

NAME

diff – differential file comparator

SYNOPSIS

diff [–efbh] file1 file2

DESCRIPTION

diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is –, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

n1 **a** *n3,n4*
n1,n2 **d** *n3*
n1,n2 **c** *n3,n4*

These lines resemble **ed** commands used to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by using **a** instead of **d** and reading backward, you can see how to convert *file2* into *file1*. As in **ed**, identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

OPTIONS

–b Causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

–e Produces a script of *a, c*, and *d* commands for the editor **ed**, which recreates *file2* from *file1*.

–f Produces a similar script, not useful with **ed**, in the opposite order. In connection with **–e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version **ed** scripts ($2,$3,...) made by A ''latest version'' appears on the standard output.

```
(shift; cat $*; echo '1,$p')| ed – $1
```

Except in rare circumstances, **diff** finds a smallest sufficient set of file differences.

–h Does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **–e** and **–f** are unavailable with **–h**.

BUGS

Editing scripts produced under the **–e** or **–f** option are naive about creating lines consisting of a single period (.).

WARNINGS

Missing newline at end of file X
indicates that the last line of file X did not have a new-line. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

FILES

/tmp/d?????
/usr/lib/diffh for **−h**

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

SEE ALSO

bdiff(1), cmp(1), comm(1), ed(1).

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [–ex3] *file1 file2 file3*

DESCRIPTION

diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	*file1* is different
====2	*file2* is different
====3	*file3* is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

f : *n1* a	Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.
f : *n1* , *n2* c	Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n*2, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a c indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

OPTIONS

–e Publishes a script for the editor **ed** that incorporates into *file1* all changes between *file2* and *file3*. That is, the changes that normally would be flagged ==== and ====3.

–x(–3) Produces a script to incorporate only changes flagged ==== (====3). The following command applies the resulting script to *file1*.

```
(cat script; echo '1,$p') | ed – file1
```

FILES

/tmp/d3*
/usr/lib/diff3prog

BUGS

Text lines that consist of a single . will defeat –e.
Files longer than 64K bytes will not work.

SEE ALSO

diff(1).

NAME

dircmp – directory comparison

SYNOPSIS

dircmp [**–d**] [–s] [–w*n*] *dir1 dir2*

DESCRIPTION

dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. It generates lists of files that are unique to each directory for all the options. If no option is entered, a list is output indicating whether the file names common to both directories have the same contents.

OPTIONS

–d	Compares the contents of files with the same name in both directories and outputs a list telling what must be changed in the two files to bring them into agreement. The list format is described in **diff**(1).
–s	Suppresses messages about identical files.
–w*n*	Changes the width of the output line to *n* characters. The default width is 72.

SEE ALSO

cmp(1), diff(1).

NAME

basename, **dirname** – deliver portions of path names

SYNOPSIS

basename *string* [*suffix*]
dirname *string*

DESCRIPTION

basename deletes any prefix ending in / and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside substitution marks (‘ ‘) within shell procedures.

dirname delivers all but the last level of the path name in *string*.

EXAMPLES

The following example, invoked with the argument /usr/src/**cmd/cat.c**, compiles the named file and moves the output to a file named **cat** in the current directory:

```
cc $1
mv a.out ‘basename $1 '\.c'‘
```

The following example sets the shell variable **NAME** to /**usr/src/cmd**:

```
NAME=‘dirname /usr/src/cmd/cat.c‘
```

SEE ALSO

sh(1).

NAME

enable, disable – enable/disable LP printers

SYNOPSIS

enable *printers*
disable [–c] [–r[*reason*]] *printers*

DESCRIPTION

enable activates the named *printers*, enabling them to print requests taken by **lp**(1). Use **lpstat**(1) to find the status of printers.

disable deactivates the named *printers*, disabling them from printing requests taken by **lp**(1). By default, any requests that are currently printing on the designated printers are reprinted in their entirety either on the same printer or on another member of the same class. Use **lpstat**(1) to find the status of printers.

OPTIONS FOR DISABLE ONLY

–c	Cancels any requests that are currently printing on any of the designated printers.
–r[*reason*]	Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next **–r** option. If the **–r** option is not present or the **–r** option is given without a reason, a default reason is used. *Reason* is reported by **lpstat**(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).

NAME

dlty – delete a type

SYNOPSIS

dlty [*options*] *type_name*

DESCRIPTION

dlty deletes a type and any installed type manager.

type_name (required) Specify the name of the type to be deleted.

OPTIONS

–n *node_spec*	Specify the node on which the type is to be deleted. You may also specify the entry directory of a volume mounted for software updates, as shown in the example below. If you omit the **–n** *node-spec* the type is deleted on the current node.
–l	List the type name/type identifier pair that is deleted.

EXAMPLES

```
$ dlty example_type -l
"example_type" 24BF9F41.100001FB deleted.

$ dlty example_type -n //test_vol -l
"example_type" 24BFA6F8.200001FB
deleted from volume //test_vol.
```

In the following example, the disk has been mounted for software updates. The disk's top level directory (cataloged as /**mount_disk** by the /etc/**mount**(1M) command) must contain a "sys" directory. If it does not, you get a "types file not found" error.

```
$ mtvol w /mount_disk
$ dlty example_type -n /mount_disk -l
 "example_type" 24BFB71E.200001FB deleted
                from volume //my_node/mount_disk.
```

SEE ALSO

crty(1), inty(1), lty(1), mount(1M)

NAME

DM commands - Display Manager commands

DESCRIPTION

Following is a list of DM commands sorted by function.

CURSOR CONTROL COMMANDS:

al	Move cursor left 1 character position.
ar	Move cursor right 1 character position.
ad	Move cursor down 1 line.
au	Move cursor up one line.
as x y	Set scale factors for arrow keys, in raster units.
curs [–on|–off]	Enable/disable cursor positioning via tn.
tl	Move cursor to the left edge of the pad.
tr	Move cursor to the end of the line.
tt	Move cursor to top edge of the window.
	tb Move cursor to the last line in the window.
twb {–l|–r|–t|–b}	Move cursor to the specified window border.
th	Move cursor right to the next horizontal tab stop.
thl	Move cursor left to the next horizontal tab stop.
	ts n1 n2...[**–r**] Set tab stops in columns n1, n2, etc., optionally repeating the last interval.
tdm	Move cursor to the Display Manager's input window.
tlw	Move cursor to the previous window.
tn	Move cursor to the next window on the display.
tni	Move cursor to next unobscured icon on the display.
ti	Move cursor to the next window in which input is enabled.

PROCESS CREATION COMMANDS:

cp [**–i**\|**–c** *char*] *pathname* [**–n** *process_name*] [*args*...]]

Create a new process, input and transcript pads, and associated windows; the process executes *pathname*; **–i** makes the window an icon; **–c** specifies the icon character; **–n** names the process.

cpo *pathname* [–n *process_name* [*args*...]]
: Create a process and execute *pathname*; do not create pads or windows.

cps pathname [–n *process_name* [*args*...]]
: Like cpo, except assign the process the SID 'user.server.none'.

PROCESS CONTROL COMMANDS:

dq [–s|–b|–c *nn*]
: Cause a quit fault, which normally terminates program execution; **–s** also stops the process; **–b** blasts the process, **–c** generates an arbitrary asynchronous fault with the specified hex status code.

ds
: Suspend execution of the process.

dc
: Continue execution of a suspended process.

WINDOW/PAD CREATION COMMANDS:

ce *pathname*
: Create an edit pad and associated window.

cv *pathname*
: Create a view, that is, a read only edit pad.

cc
: Create a copy of an existing window.

WINDOW CONTROL COMMANDS:

wg
: Grow or shrink a window.

wge
: Grow or shrink a window with feedback.

wm
: Move a window.

wme
: Move a window with feedback.

wp [window_name|group_name] [–t|–b]
: Push (named) window (or window group) to bottom of pile if unobscured, else pop to top. **–t** and **–b** will force a window to the bottom or to the top.

wc [–f] [–q] [–a] [–s]
: Close (delete) a window. Use **–a** to automatically close and delete a window after a ^Z and **–s** to reverse auto-close mode.

wa [**–on|–off**]
: Toggle auto-hold mode.

ws [**–on|–off**]
: Toggle window-at-a-time scroll mode.

wh [–on|–off]
: Toggle hold mode.

wdf [*n*]
: Set the *n*'th default window creation boundaries.

PAD CONTROL COMMANDS:

pb
: Move the bottom of the pad into window.

pt	Move the top of the pad into window.
pp [–]*n*	Move the pad forward [**backward**] *n* pages (*n* may be decimal fraction).
pv [–]*n*	Move the pad forward [**backward**] *n* lines (*n* may be decimal fraction).
ph [–]n	Move the pad *n* character positions horizontally (*n* may be decimal fraction).
pn *pathname*	Save the pad under *pathname* (transcript pads only).

WINDOW GROUP AND ICON COMMANDS:

icon [entry_name] [–i] [–w] [–c 'char']	Make a window or group of windows into an icon(s).
wi [entry_name] [–i] [–w]	Make a window or group of windows invisible.
wgra *group_name* **[entry_name]**	Add a window to a group of windows.
wgrr *group_name* **[entry_name]**	Remove a window from a group of windows.
cpb *group_name*	Display a list of the windows in a group.
idf	Set icon positioning vector.

PAD EDITING COMMANDS:

Set modes:

ro [–on|–off]	Change pad from write to read-only mode or vice versa.
ei [–on|–off]	Change from insert to overstrike mode or vice versa.

Insert text:

es 'string'	Insert 'string' at the current cursor position.
en	Insert a new line character.
er nn	Send raw hexadecimal data byte *nn* to user program.
eef	Insert a stream end-of-file indicator.

Delete text:

ed	Delete the character at the cursor position.
ee	Delete the character immediately preceding the cursor.

Cut and Paste:

xc [–r] [–f pathname | *name*]	Copy text into a paste buffer or file.
xd [–r] [–f *pathname* | name]	Copy text into a paste buffer or file and delete text.

xp [–r] [–f *pathname* | name]
: Insert contents of paste buffer or file into pad.

xi [–f *pathname*]
: Copy display image to graphics map file for above cut and paste commands: use **–r** for a rectangular cut. use **–f** to specify a file name.

Search:

/regular exp/	Search forward in the pad for a string which matches the regular expression; for help on regular expressions, type help patterns.
\regular exp\	Search backward in the pad for a string which matches the regular expression.
// or \\	Repeat last search forward or backward.
sq	Abort search.
sc [**–on**|**–off**]	Enable/disable case sensitivity for searches.

Substitute:

s/re/replace/	Substitute the replacement text for all strings in the range which match the regular expression .
so/re/replace/	Substitute the replacement text for the first string in each line in the range which matches the regular expression.

Miscellaneous:

undo	Undo file changes in an input pad or an edit pad; successive undos will undo further back in history.
pw	Write edit pad to new file, but don't close pad or delete window.
echo [–r]	If a grow/move is in progress, then end feedback. Else begin text highlighting feedback if the cursor is on text.
abrt	Abort text or window feedback, abort a search, or clear mark stack.

case [–u|–l|–s]
: Change the case of the letters in a marked text range.

–s	Switch to inverse case (default)
–u	Change to upper case
–l	Change to lower case

KEY DEFINITION COMMANDS:

kd key [[def] ke]
: Set or display a DM key definition.

kbd n
: Declare keyboard type; *n* must be '3' if your node has a Domain Programmable Keyboard (with numeric keypad); *n* must be '2' if your node has a Low Profile Keyboard; *n* must be ' ' if your node has an 880 keyboard; this command is only valid during node boot.

&'prompt'	Write the optional prompt string in the input pad, then read a line of input

DISPLAY MANAGEMENT COMMANDS:

Login/Logout:

l pers [group [org]]	Login (valid only when logged out); the 'l' is optional when preceded by the "login:" prompt.
lo [−f]	Logout (valid only when logged in)
ex	Exit DM to boot shell.
shut [−f]	Shutdown the system

Place/Clear Marks:

dr	Place a mark (for window control or cut and paste).
gm	Go to mark.
cms	Clear mark stack.
rm	Push last mark back on the stack.

Miscellaneous:

=	Display line, column number, and x,y pixel values of current cursor position.
aa	Acknowledge DM alarm.
ap	Acknowledge DM alarm and pop the associated window .
bl [−i|−c] [*l_char*] [*r_char*]	Check and/or balance delimiting characters.
env var [*value*]	Set or display an environment variable; setting an environment variable is only valid during login startup.
bgc [−on|−off]	Turn on or off the gray scale background color (monochrome monitor).
inv [-on|−off]	Invert the screen to black on white, or vice-versa (monochrome monitor).
mono [−on|−off]	Enable/disable monochrome mode (color monitor).
msg 'string'	Display a message in the DM output window.

rs	Refresh the entire screen.
rw [**−r**]	Refresh current window; **−r** option reenables window.
fl *pathname* [**−i**]	Load a font to be used in later pads; **−i** indicates an icon font.
cmdf *pathname*	Execute DM script.

NAME

dspst – display process status graphically

SYNOPSIS

dspst [**–r** *n*] [**–p**] [**–L1**] [**–os**] [**–m**]
[**–io**] [**–a**] [**–n** *node_spec*]
[**–large**|**–small**]

DESCRIPTION

dspst displays process statistics in a graphical, bar-chart fashion within the current process window. The chart is updated periodically (see **–r** below). The default action of this command is to display the brief Domain/OS process list, all user processes and all I/O information in a font size automatically selected based on window size.

While **dspst** is running, the following keys are interpreted as follows:

All Keyboards:

CRTL/T	Move to top
CRTL/B	Move to bottom
RETURN	Exit
CRTL/N	Exit
CRTL/Y	Exit and save current image
Boxed up arrow	Scroll backward 1/2 window
Boxed down arrow	Scroll forward 1/2 window
Shifted up arrow	Scroll backward 1 line
Shifted down arrow	Scroll forward 1 line
EXIT or ABORT	Exit
SAVE	Exit and save current image

OPTIONS

–r *n*	Specify that the display should be repeatedly updated every *n* seconds. If this option is omitted, the display is updated every 4 seconds.
–p	Show process information.
–l1	Show Domain/OS and user-process information.
–os (default)	Show brief Domain/OS and full user-process information.
–m	Show missing CPU time.
–io (default)	Show I/O statistics.
–a	Show all information (same as **–l1** **–io** **–m**).
–n *node_spec*	Specify remote node whose process statistics are to be listed.

–large (default)	Force use of large font for display.
–small	Force use of small font for display.

EXAMPLES

1. Display Domain/OS, user process, and I/O status.

```
$ dspst
```

2. Display Domain/OS, user process, and I/O status for the node named //**fred** using the large font.

```
$ dspst –n //fred –large
```

NAME

du – summarize disk usage

SYNOPSIS

du [–ars] [*names*]

DESCRIPTION

du prints the number of blocks (1024 bytes per block) contained in all files and directories specified by the *names* argument. The block count includes the indirect blocks of the file. A file with two or more links is only counted once. If the *names* argument is missing, a period (.) is used.

OPTIONS

–a	Generates an entry for each file.
–r	Generates messages about such things as directories that cannot be read and files that cannot be opened. **du** is normally silent about these things.
–s	Prints only the grand total of blocks for each of the specified *names*.

BUGS

Absence of the **–a** or –s options causes an entry to be generated for each directory only.

If the **–a** option is not used, nondirectories given as arguments are not listed.

If too many distinct linked files exist, **du** counts the excess files more than once.

Files with holes in them will get an incorrect block count.

NAME

dump – dump selected parts of an object file

SYNOPSIS

dump [*options*] *files*

DESCRIPTION

The **dump** command dumps selected parts of each of its object *file* arguments.

dump accepts both object files and archives of object files. It processes each file argument according to one or more of the following *options*.

OPTIONS

–a	Dumps the archive header of each member of each archive file argument.
–c	Dumps the string table.
–f	Dumps each file header.
–g	Dumps the global symbols in the symbol table of an archive.
–h	Dumps section headers.
–l	Dumps line number information.
–L	Interprets and prints the contents of the *.lib* sections.
–o	Dumps each optional header.
–r	Dumps relocation information.
–s	Dumps section contents.
–t	Dumps symbol table entries.
–z *name*	Dumps line number entries for the named function.
–Aa	Dumps the longname table and module table of an archive.
–Ai	Interprets and prints the contents of the *.inlib* section.
–Am	Interprets and prints the *.mir* section records.
–Ar	Interprets and prints the *.rwdi* section records.
–AR	Interprets and prints the contents of the *.rwdi* section.
–As	Interprest and prints the *.sri* section records.
–AS	Interprets and prints section contents.

MODIFIERS

The following *modifiers* are used in conjunction with the *options* listed above to modify their capabilities.

–d *number* Dumps the section number, *number*, or the range of sections starting at *number* and ending at the *number* specified by **+d**.

+d *number* Dumps sections in the range either beginning with first section or beginning with section specified by **−d**.

−n *name* Dumps information pertaining only to the named entity. This *modifier* applies to **−h**, −s, **−r**, **−l**, and **−t**.

−p Suppresses printing of the headers.

−t *index* Dumps only the indexed symbol table entry. The **−t** used in conjunction with **+t**, specifies a range of symbol table entries.

+t *index* Dumps the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **−t** *option*.

−u Underlines the name of the file for emphasis.

−v Dumps information in symbolic representation rather than numeric (e.g., C_STATIC instead of **0X02**). This *modifier* can be used with all the above *options* except −s and −o *options* of **dump**.

−z *name,number*
Dumps line number entry or range of line numbers starting at *number* for the named function.

+z *number* Dumps line numbers starting at either function *name* or *number* specified by −z, up to *number* specified by +z.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the −z *option* may be replaced by a blank.

dump attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

SEE ALSO

a.out(4), ar(4).

NAME

echo – echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	Backspace
\c	Print line without new-line
\f	Formfeed
\n	Newline
\r	Carriage return
\t	Tab
\v	Vertical tab
\\	Backslash
\0*n*	Where *n* is the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number representing that character.

echo is useful for producing diagnostics in command files and for sending known data into a pipe.

CAVEATS

When representing an 8-bit character by using the escape convention **\0***n*, the *n* must always be preceded by the digit zero (0).

For example, typing: **echo 'WARNING:\07'** will print the phrase **WARNING:** and sound the "bell" on your terminal. The use of single (or double) quotes (or two backslashes) is required to protect the "\" that precedes the "07".

For the octal equivalents of each character, see ascii(5), in the *SysV Programmer's Reference*.

SEE ALSO

sh(1).

NAME

ed, red – text editor

SYNOPSIS

ed [**–s**] [**–p** *string*] [*file*]

red [**–s**] [**–p** *string*] [*file*]

DESCRIPTION

ed is the standard text editor. If the *file* argument is given, **ed** simulates an *e* command (see below) on the named file; that is, the file is read into **ed**'s buffer so that it can be edited.

ed operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

red is a restricted version of **ed**. It only allows editing of files in the current directory. It prohibits executing shell commands using the *!shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both **ed** and **red** support the **fspec**(4) formatting capability. After including a format specification as the first line of *file* and invoking **ed** with your terminal in **stty –tabs** or **stty tab3** mode (see **stty**(1)), specified tab stops are automatically used when scanning *file*. For example, if the first line of a file contained:

```
<:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: while inputing text, tab characters when typed are expanded to every eighth column as is the default.

OPTIONS

–s Suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a *!shell command*. Also, see the **WARNING** section at the end of this manual page.

–p Allows you to specify a prompt string. Commands to **ed** have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While **ed** is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line, followed immediately by a carriage return.

ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (*s*, for example) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE.

REGULAR EXPRESSIONS

The following *one-character RE*s match a *single* character:

- An ordinary character (*not* one of those discussed below) is a one-character RE that matches itself.
- A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:
  - ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets.
  - ^ (caret or circumflex), which is special at the *beginning of an entire* RE, or immediately follows the left of a pair of square brackets.
  - $ (dollar sign), which is special at the *end* of an entire RE.
  - The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the g command, below.)
- A period (.) is a one-character RE that matches any character except new-line.
- A non-empty string of characters enclosed in square brackets ([ ]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (–) may be used to indicate a range of consecutive ASCII characters; for example, **[0–9]** is equivalent to **[0123456789]**. The – loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., [ ]a–f] matches either a right square bracket (]) or one of the letters a through f inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *RE*s from one-character REs:

- A one-character RE is a RE that matches whatever the one-character RE matches.
- A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

- A one-character RE followed by \{*m*\}, \{*m,*\}, or \{*m,n*\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m,*\} matches *at least m* occurrences; \{*m,n*\} matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.
- The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.
- A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.
- The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) the sub-expression specified is that beginning with the *n*–th occurrence of \(counting from the left. For example, the expression **^\(.*\)\1$** matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both).

- A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.
- A dollar sign ($) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction ^*entire RE* $ constrains the entire RE to match the entire line.

The null RE (e.g., //) is equivalent to the last RE encountered. See also the last paragraph before **FILES** below.

To understand addressing in ed it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

– The character **.** addresses the current line.

– The character **$** addresses the last line of the buffer.

– A decimal number *n* addresses the *n*-th line of the buffer.

– *'x* addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.

– A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before **FILES** below.

- A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.
- An address followed by a plus sign (+) or a minus sign (–) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
- If an address begins with + or –, the addition or subtraction is taken with respect to the current line; e.g, –5 is understood to mean .–5.
- If an address ends with + or –, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and the rule immediately above, the address – refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to –.) Moreover, trailing + and – characters have a cumulative effect, so –– refers to the current line less 2.
- For convenience, a comma (,) stands for the address pair **1,$**, while a semicolon (;) stands for the pair **.,$**.

COMMANDS

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches. The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of **ed** commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n*, and *p* commands.

(.)**a**
<text>

The *a*ppend command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the ''appended'' text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*

The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a shell (sh(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also **DIAGNOSTICS** below.

E *file*

The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file*

If *file* is given, the *f*ile-name command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1,$)g/*RE*/*command list*

In the *g*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted. The . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also **BUGS** and the last paragraph before **FILES** below.

(1 , $)G/*RE*/

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an & causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *h*elp command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The *H*elp command causes ed to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous ? if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(., .+1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. The address '*x* then addresses this line; . is unchanged.

(., .)l

The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab, backspace*) are represented by visually mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(., .)m*a*

The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(.,.)n

The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p

The *p*rint command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*. For example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done; however, see **DIAGNOSTICS**, below.

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

($)r ***file***

The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (sh(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE*/*replacement*/ or
(.,.)s/*RE*/*replacement*/g or
(.,.)s/*RE*/*replacement*/n n = 1-512

The *s*ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator g appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number n appears after the command, only the n th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before **FILES** below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u

The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1,$)v/*RE*/*command list*

This command is the same as the global command *g* except that the *command list* is executed with . initially set to every line that does *not* match the RE.

(1,$)V/*RE*/

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the RE.

(1,$)w *file*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see **umask**(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of c haracters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell (**sh**(1)) command whose standard input is the addressed lines. Such a shellcommand is *not* remembered as the current file name.

($)=

The line number of the addressed line is typed; . is unchanged by this command.

!*shell command*

The remainder of the line after the ! is sent to the UNIX system shell (**sh**(1)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(.+1)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, **ed** prints a ? and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, and 64 characters per file name. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, **ed** discards ASCII NUL characters. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by **ed**.

If a file is not terminated by a new-line character, **ed** adds one and outputs a message explaining what it did.

If the closing delimiter of a RE or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

```
s/s1/s2     s/s1/s2/p
g/s1        g/s1/p
?s1         ?s1?
```

WARNINGS

The – *option*, although supported in this release for upward compatibility, will no longer be supported in the next major release of the system. Convert shell scripts that use the – *option* to use the –s *option*, instead.

BUGS

A *!* command cannot be subject to a *g* or a *v* command.
The *!* command and the ! escape from the *e*, *r*, and *w* commands cannot be used if the editor is invoked from a restricted shell (see **sh**(1)).
The sequence **\n** in a RE does not match a new-line character.

Characters are masked to 7 bits on input.
If the editor input is coming from a command file (e.g., ed file < ed-cmd-file), the editor will exit at the first failure.

FILES

/usr/tmp	Default directory for temporary work file.
$TMPDIR	If this environmental variable is not null, its value is used in place of **/usr/tmp** as the directory name for the temporary work file.
ed.hup	Work is saved here if the terminal is hung up.

DIAGNOSTICS

?	Command error.
? file	An inaccessible file.
	(use the **help** command for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, **ed** warns the user if an attempt is made to destroy **ed**'s buffer via the *e* or *q* commands. It prints ? and allows one to continue editing. A second *e* or *q* command at this point will take effect. The –s command-line **option** inhibits this feature.

SEE ALSO

edit(1), ex(1), grep(1), sed(1), sh(1), stty(1), umask(1), vi(1).
fspec(4), regexp(5) in the *SysV Programmer's Reference*.

NAME

edfont – edit a character font

SYNOPSIS

edfont [*file* | –v]

DESCRIPTION

edfont is an interactive program with both menu-driven and command-line interfaces. It allows you to create, edit, and view character font files. You can specify the font file with the **file** parameter, or use the "Open Font" entry in the "File" menu. If the –v option is used, **edfont** will print its version number and exit.

Generally, you must press the left mouse button <M1> to activate commands in the menu-driven interface. When you must enter a string (for example, when you designate which font you want to open) and there is a "Done" field on the menu, enter the string, point to "Done" and press <M1> to activate. If "Done" does not appear when you must enter a string, simply type the string and press <RETURN> to activate the command.

When using the menu-driven interface, you may notice that you cannot always select every menu choice. For example, you can't select "Open Font" if you already have one open, and likewise it's invalid to try to close a font when no font is open. When commands are invalid, as in these cases, their places on the menus are grayed out so that they can't be selected.

edfont lets you select a character (glyph) in a variety of ways. The utility interprets input this way:

- Any three-character string whose first character is a lowercase c has its final two characters interpreted as a compose sequence (e.g., ca^ for lowercase a with a circumflex accent)
- Any string that begins with 0x is interpreted as a hexadecimal code (e.g., **0x41** for uppercase A)
- Any string that begins with 0 (zero) is interpreted as octal (e.g., **0101** for A)
- Any string that begins with a digit other than zero is considered to be decimal (e.g., **65** for A)
- Any other string is considered to be an ASCII character (e.g., A for A)

For more information on compose sequences, see your system's User's Guide. For a list of decimal, octal, and hexadecimal values for the characters in Apollo's default character set, as well as a list of the compose sequences, see the files in the /usr/**pub** directory.

When you invoke **edfont**, it sets default values for several variables. You can change those defaults using either the appropriate command in the menu-driven interface or set in the command-driven interface. For more information on these interfaces see the section on command interfaces, below.

The following table lists variables, their types, default values (if any), and purpose.

Variable/Type	Default	Description
fontpath/string	**:/sys/dm/fonts**	List of directories, separated by colons, in which **edfont** should search for fonts
fontservers/string	**/usr/apollo/lib/edfont**	The search path for the font servers directory
fill/string	**outline**	The name of the current fill pattern
fontorigin/coord	none	The coordinate value that tells the number of pixels below and to the left of the font origin
fontsize/coord	none	The width and height of the font bounding box
fontspacing/coord	none	The horizontal and vertical font spacing (leading)
glyphoffset/coord	none	The offset of the current glyph from the font origin
glyphsize/coord	none	The width and height of the bitmap for the current glyph
glyphwidth/coord	none	The number of pixels from the right edge of the current glyph to the left edge of the next glyph
mask/string	**src ^ dst**	The current mask (raster operation)

edfont handles fonts created using Apollo's current and pre-SR10 formats, as well as Adobe BDF fonts.

Menu Interface

Note: You can get additional information about any item on the display by pressing the HELP key at the cursor position where you need help. This pops a help box. To return to the original display, move the cursor out of the help box.

Font When you position the cursor here and press <M1>, **edfont** displays a menu with the following choices:

Open Font
Close Font
Select Glyph
Font Params
Glyph Params
Quit

Use these choices to open and close the font you want to edit, select an individual glyph (character) to edit, and examine or change the font's parameters or a single glyph's parameters.

Tools If you press <M1>, you will see the following choices:

Grid
Metrics

By default, both are turned on. If you turn off Grid, you no longer will see the pixel-by-pixel bitmap grid in the edit window. If you turn off Metrics, the glyph fills the edit window.

Metrics shows these three attributes of your glyph and font:

- Origin and baseline (fine dotted line)
- Glyph-bounding box (long dashed line)
- Font-bounding box (short dashed line)

Commands If you press <M1>, you will see the following choices:

Undo	Undo remembers your last 10 changes to the current glyph. Undo does not work on parameter changes, however.
Run Commands	You can set up a file of commands and direct **edfont** to execute that file. For more information on the commands you can use, see the description of the command interface, below.
Copy Glyph	Copies a glyph from elsewhere in your font or from another font.
Delete Glyph	This deletes a glyph.
Rotate Glyph	This rotates a glyph by the number of degrees you specify.

Draw When you position the cursor here and press <M1>, you will see the following choices:

Pixel	Manipulate individual pixels
Freehand	Draw freehand
Line	Draw lines
Box	Draw boxes
Circle	Draw circles
Cut	Select and delete a pixel or range of pixels
Copy	Select and copy a pixel or range of pixels

Paste	Paste in a pixel or range of pixels that you have previously cut or copied
Zoom	Zoom in on a selected portion of the glyph

Note that after you Cut or Copy, **edfont** automatically changes the Draw mode to Paste. You can manually change it to something else if you prefer.

Fill When you position the cursor here and press <M1>, you will see the following choices:

Outline	this is the default
25% gray	
50% gray	
75% gray	
black	
bricks	
chex	
/stripes	right-leaning stripes
\stripes	left-leaning stripes
|stripes	vertical stripes
–stripes	horizontal stripes
tri	
waves	

The way **edfont** fills an entity such as a circle or box depends on which fill you choose. If you choose 50% gray, for example, and then create a box, **edfont** turns on half of the pixels inside the box to create a 50% gray effect. If you choose 75% or 25% gray, **edfont** turns on proportionally more or fewer pixels to get the desired effect.

Mask When you position the cursor here and press <M1>, you will see the following choices (where "src" means source, "dst" means destination, and the other characters are logical operators):

Menu Choices	Logical Operation
clear	Assign zero to all new destination values
src & dst	Assign source AND destination to new destination
src & ~dst	Assign source AND complement of destination to new destination
src	Assign source values to new destination
~src & dst	Assign complement of source AND destination to new destination

dst	Assign all destination values to new destination
src ^ dst	Assign source EXCLUSIVE OR destination to new destination (default)
src | dst	Assign source OR destination to new destination
~(src | dst)	Assign complement of source AND complement of destination to new destination
src == dst	Assign source EQUIVALENCE destination to new destination
~dst	Assign complement of destination to new destination
src | ~dst	Assign source OR complement of destination to new destination
~src	Assign complement of source to new destination
~src | dst	Assign complement of source OR destination to new destination
~(src & dst)	Assign complement of source OR complement of destination to new destination
set	Assign 1 to all new destination values

Setting the mask value turns pixels on. That is, if you select a pixel or range of pixels with this mask, all the pixels turn black, regardless of whether they already were black. The mask clear turns a pixel or range of pixels off (white), regardless of the pixel's initial value.

The default mask src ^ dst toggles pixels. That is, if they already were black, they become white, and vice versa. However, if you are drawing in Freehand mode, this mask toggles the first pixel you cross and then sets the rest of the pixels you cross to that first pixel's value.

When you have a font open, the menu-driven interface also includes two boxes on the right side of the display labeled "<<<" and ">>>". The two are for displaying the previous and next glyph, respectively, in the current font. Move the cursor over either box and press <M1> to activate.

Command Interface

In addition to **edfont**'s menu-driven interface, you can use the following commands in the input pad at the bottom of the **edfont** window, or embed them in **edfont** scripts.

Commands (Arguments)	Description
!*shell-command*	Run a shell command in the **edfont** window.
box *x1 y1 x2 y2*	Draw a box that is bounded by (*x1,y1*) and (*x2,y2*).
circle *x y r*	Draw a circle which has its center at (*x,y*) and a radius of *r*.

close [–save|–nosave]	Close the font. If you specify **–save**, **edfont** saves your changes, while if you specify **–nosave**, **edfont** ignores them.
copy *glyphcode* [*fontfile*]	Copy the specified glyph to the current glyph. If you specify a *fontfile*, **edfont** copies the glyph from that font; otherwise, it copies the glyph from the current font.
delete	Delete the current glyph.
grid on | off	Turn the bitmap grid **on** or **off**.
help [*command*]	Get a list of available commands, or get help on the specified command.
line *x1 y1 x2 y2*	Draw a line that begins at (*x1,y1*) and ends at (*x2,y2*).
metrics on | off	Turn the font metrics display **on** or **off**.
next	Go to the next glyph in the current font.
open *fontfile*	Open the specified fontfile.
pixel *x y*	Draw a pixel at (*x,y*).
previous	Go to the previous glyph in the current font.
quit [–save|–nosave]	Exit **edfont**, closing the current font (if one is open). See **close** for information on **–save** and **–nosave**.
Commands (Arguments)	Description
rotate *degrees*	Rotate the current glyph by the specified number of degrees.
select *glyphcode*	Go to the specified glyph. For information on entering a glyph or *glyphcode* see the Description section above.
set *var=value*	Set *var* to the specified value. *var* can be one of the **edfont**'s parameters, as described in the Description section above.
source *filename*	Execute the command-script *filename*.
undo	Undo the last bitmap operation.
unzoom	Zoom out one level.
zoom *x1 y1 x2 y2*	Zoom in so that the view is filled with the box bounded by (*x1, y1*) and (*x2,y2*).

NAME

edit – text editor (variant of ex for casual users)

SYNOPSIS

edit [**–r**] *name* ...

DESCRIPTION

edit is a variant of the text editor **ex** recommended for new or casual users who wish to use a command-oriented editor.

OPTION

–r Recovers file after an editor or system crash. The following brief introduction should help you get started with **edit**. If you are using a CRT terminal you may want to learn about the display editor **vi**.

To edit the contents of an existing file you begin with the command "edit name" to the shell. **edit** makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run **edit** on it; you will cause an error diagnostic, but do not worry.

edit prompts for commands with the character ':', which you should see after starting the editor. If you are editing an existing file, then you will have some lines in **edit**'s buffer (its name for the copy of the file you are editing). Most commands to **edit** use its "current line" if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and hit carriage return (as you should after all **edit** commands) this current line will be printed. If you **delete** (**d**) the current line, **edit** prints the new current line. When you start editing, **edit** makes the last line of the file the current line. If you **delete** this last line, then the new last line becomes the current one. In general, after a **delete**, the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you give this command (typing a carriage return after the word **append**) **edit** will read lines from your terminal until you give a line consisting of just a ".", placing these lines after the current line. The last line you type then becomes the current line. The command **insert** (**i**) is like **append** but places the lines you give before, rather than after, the current line.

edit numbers the lines in the buffer, with the first line having number 1. If you give the command "1" then **edit** types this first line. If you then give the command **delete edit** deletes the first line, line 2 will become line 1, and **edit** prints the current line (the new line 1) so you can see where you are. In general, the current line is always the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (s) command. Use s/**old**/**new**/ where *old* is replaced by the old characters you want to get rid of and *new* is the new characters you want to replace it with.

The command **file** (f) tells you how many lines there are in the buffer you are editing and will say ``[Modified]'' if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a **write** (w) command. You can then leave the editor by issuing a **quit** (q) command. If you run **edit** on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from **edit** after modifying the buffer without writing it out, you are warned that there has been ``No **write** since last change'' and **edit** awaits another command. If you wish not to **write** the buffer out then you can issue another **quit** command. The buffer is then irretrievably discarded, and you return to the shell.

By using the **delete** and **append** commands, and giving line numbers to see lines in the file you can make any changes you desire. You should learn at least a few more things, however, if you are to use **edit** more than a few times.

The **change** (c) command changes the current line to a sequence of lines you supply (as in **append** you give lines up to a line consisting of only a ``.''). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e., ``3,5change''. You can print lines this way too. Thus ``1,23p'' prints the first 23 lines of the file.

The **undo** (u) command reverses the effect of the last command you gave which changed the buffer. Thus if you give a **substitute** command which does not do what you want, you can say **undo** and the old contents of the line will be restored. You can also **undo** an **undo** command so that you can continue to change your mind. **edit** gives you a warning message when commands you do affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an *undo* and looking to see what happened. If you decide that the change is ok, then you can *undo* again to get it back. Note that commands such as *write* and *quit* cannot be undone.

To look at the next line in the buffer you can just hit carriage return. To look at a number of lines hit ^D (control key and, while it is held down D key, then let up both) rather than carriage return. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command ``z.''. The current line will then be the last line printed; you can get back to the line where you were before the ``z.'' command by saying ``\*(rq'. The z command can also be given other following characters ``z–'' prints a screen of text (or 24 lines) ending where you are; ``z+'' prints the next screenful. If you want less than a screenful of lines, type in "z.12" to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command ``delete 5''.

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form /text/ to search forward for *text* or ?text? to search backward for *text*. If a search reaches the end of the file without finding the text it wraps, end around, and continues to search back to the line where you are. A useful feature here is a search of the form **/^text/** which searches for *text* at the beginning of a line. Similarly **/text$/** searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has a symbolic name "."; this is most useful in a range of lines as in ".,$print" which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name "$". Thus the command "$ delete" or "$d" deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line "$–5" is the fifth before the last, and ".+20" is 20 lines after the present.

You can find out which line you are at by doing ".=". This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move you can then say "10,20delete a" which deletes these lines from the file and places them in a buffer named **a.edit** and has 26 such buffers named *a* through *z*. You can later get these lines back by doing "put a" to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files you can give an **edit** (e) command after copying the lines, following it with the name of the other file you wish to edit, i.e., "edit chapter2". By changing *delete* to *yank* above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say "10,20move $" for example. It is not necessary to use named buffers in this case (but you can if you wish).

SEE ALSO

ed(1), ex(1), vi(1).

NAME

edmtdesc – edit magtape descriptor file

SYNOPSIS

edmtdesc {*options*} *pathname*

DESCRIPTION

edmtdesc allows you to create, list, and modify the magnetic tape descriptor object. The descriptor file provides information to the streams manager so that it can handle subsequent tape operations.

pathname (required)	Specify name of magtape descriptor file to be created, listed, or edited.

OPTIONS

At least one of the following options must be specified.

–c	Create a new magtape descriptor object with the name given in the *pathname* argument.
–l [*var*...]	List the values of the variable(s) specified. If no variables are named, the entire magtape descriptor is listed.
–s {*var value*}...	Set the variable(s) indicated to the specified value(s). At least one variable/value pair is required if –s is specified. Multiple variable/value pairs are permitted, separated by blanks.

Variables

The variables known to **edmtdesc** are listed below, along with their types and default values. The variable types are: integer (int), Boolean (y/n), character string of *n* letters (c [*n*]), and date (in format yy/mm/dd.hh:mm).

Name	Type	Default	Definition
dev	c[1]	m	Device type ('m' for magtape, 'c' for cartridge)
u	int	0	Magtape unit number (normally 0)
lab	y/n	yes	'Yes' if magtape is ANSI labeled, 'no' if unlabeled
reo	y/n	no	'Yes' to reopen previously used volume, 'no' to open new volume ('yes' suppresses rewind)
clv	y/n	yes	'Yes' closes volume when file is closed, 'no' leaves volume open

Name	Type	Default	Definition
spos	y/n	no	'Yes' saves volume position when volume is closed (for reopen), 'no' rewinds volume when closed
vid	c[6]	–auto	Volume identifier (labeled volumes)
vacc	c[1]		Volume accessibility (labeled volumes)
own	c[14]	–auto	Volume owner (labeled volumes)
f	int*	1	file sequence number: integer or "cur" for current file, or "end" for new file at end of labeled volume
rf	c[1]	D	record format -- "f" for fixed length, "d" for variable length, "s" for spanned, "u" for undefined
bl	int	2048	block length, in bytes
rl	int	2048	(maximum) record length, in bytes
ascnl	y/n	yes	'Yes' for ASCII newline handling (strip newlines on write, supply them on read), 'no' for no newline handling
fsect	int	1	File section number (labeled volumes)
fid	c[17]		File identifier (labeled volumes)
fsid	c[6]		File set identifier (labeled volumes)
gen	int	1	Generation of file (labeled volumes)
genv	int	1	Generation version of file (labeled volumes)
cdate	date	–auto	Creation date of file (labeled volumes)
edate	date	–auto	Expiration date of file (labeled volumes)
facc	c[1]		File accessibility (labeled volumes)
sysc	c[xx]		System code (labeled volumes)
sysu	c[xx]		System use (labeled volumes)
boff	int	0	Buffer offset (labeled volumes, should be 0)

For cartridge tape (**dev c**), you must change the block length (**bl**) and the record length (**rl**) to be 512 or less and the record format to be fixed ("**rf f**").

EXAMPLES

Edit file set_tape; set the tape unit number to 1; declare tape as ANSI labeled.

```
$ edmtdesc set_tape –s u 1 lab yes
```

Create descriptor file ct for cartridge tape, blocking 4 records of maximum length 128 to each block.

```
$ edmtdesc ct –c –s dev c bl 512 rl 128 rf f
```

NAME

egrep – search a file for a pattern using full regular expressions

SYNOPSIS

egrep [*options*] *full regular expression* [*file*...]

DESCRIPTION

egrep (*expression grep*) searches files for a pattern of characters and prints all lines that contain that pattern. **egrep** uses *full regular expressions* (expressions that have string values that use the full set of alphanumeric and special characters) to match the patterns. It uses a fast deterministic algorithm that sometimes needs exponential space.

egrep accepts *full regular expressions* as in **ed**(1), except for \(and \), with the addition of:

- A *full regular expression* followed by + that matches one or more occurrences of the *full regular expression.*
- A *full regular expression* followed by ? that matches 0 or 1 occurrences of the *full regular expression.*
- *Full regular expressions* separated by | or by a new-line that match strings that are matched by any of the expressions.
- A *full regular expression* that may be enclosed in parentheses ( ) for grouping.

Be careful using the characters $, *, [, ^, | , (,), and \ in *full regular expression*, because they are also meaningful to the shell. It is safest to enclose the entire *full regular expression* in single quotes ′...′.

The order of precedence of operators is [ ], then * ? +, then concatenation, then | and new-line.

If no files are specified, **egrep** assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

OPTIONS

–b	Precedes each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
–c	Prints only a count of the lines that contain the pattern.
–i	Ignores upper/lower case distinction during comparisons.
–l	Prints the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.
–n	Precedes each line by its line number in the file (first line is 1).
–v	Prints all lines except those that contain the pattern.

–e *special_expression*
Searches for a *special expression* (*full regular expression* that begins with a –).

–f *file* Takes the list of *full regular expressions* from *file*.

BUGS

Ideally there should be only one **grep** command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h.**

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

SEE ALSO

ed(1), fgrep(1), grep(1), sed(1), sh(1).

NAME

emt – emulate a dumb terminal

SYNOPSIS

emt [*pathname*]

DESCRIPTION

emt allows your node to emulate an ASCII terminal connected to another computer. This asynchronous connection exists through a stream opened on one of the node's SIO lines. **emt** also permits ASCII file transfer between your node and the remote host.

pathname (optional) Specify file containing **emt** commands.

Default if omitted: read commands from standard input

emt begins execution in local mode, and displays the following prompt:

emt>

To enter remote mode, press F1. (The **emt** command **dl** no longer exists.) In remote mode, your terminal operates as if it were physically connected to the remote computer ("host"). You can log on and enter remote host commands.

To return to local mode, press F1 again.

INPUT/OUTPUT STREAMS

emt uses the four standard streams: standard input, standard output, error input, and error output, as follows:

- **emt** commands are read from an **emt** command file or from standard input. The command filename may be specified on the command line or using the **emt run** command. Up to four levels of command files may be nested. When EOF is reached in a command file, commands are read from the previous file or from standard input. If EOF is reached on standard input, **emt** exits.
- Keystrokes to be sent to the host computer are read from standard input only.
- The **emt** command responses and all messages from the host are written to standard output.
- Error messages from Aegis system calls are written to error output. Optional monitoring (monit) may also be written to error output (or to a named file).

You may use redirection of standard input, command-line specification of a command file or the **emt run** command to automate **emt** usage and use **emt** in shell scripts. **emt** behaves slightly differently with regard to host transmissions, depending on which of these techniques you use and you may select the method that best suits your purpose.

When input is redirected to standard input ('**emt <emtfile1**'), lines in the command file that are sandwiched between **F1** commands (enter/exit remote mode) are transmitted to

the host. Other lines outside **F1** commands are interpreted and executed as **emt** commands.

Contents of **emtfile1**:

Command	**Description**
interm lf	Sets input terminator.
outterm lf	Sets output terminator.
list	Lists **emt** state settings.
F1	Invokes remote mode (communication to host).
hello host	This and succeeding lines get sent to host.
goodbye host	Last line sent to host.
~1i	**emtesc** char, specifies '**F1**', return to local mode.
list	Back in local mode, lists **emt** state settings.
q	Exit from **emt**.

When a command file is invoked either via the command line (**emt emtfile2**) or by using the **run** command (**run emtfile2**), the behavior is different in that lines following **F1** commands are not transmitted to the host. This is because host transmissions are read from standard input and standard input has not been redirected to the file:

Contents of **emtfile2**:

Command	**Description**
interm lf	Sets input terminator.
outterm lf	Sets output terminator.
list	Lists **emt** state settings.
F1	Invokes remote mode (communication to host). All host input is now taken from the keyboard (or from standard input if it has been otherwise redirected). Finally user types ~1 or presses **F1** to return to local mode.
list	Local mode, **emt** commands read from **emtfile2** again.
q	Exit from **emt**.

You may also use the **xmit** command to transmit a file (of commands or data) to the host. Use the **emt rcv** command to receive host transmissions to a Domain file.

TRANSFERRING FILES

You can transfer files using **emt's** receive (**rcv**) or transmit (**xmit**) commands. **xmit** sends a Domain file to the remote host. **rcv** opens a Domain file to receive information from the remote host. For example, if you type (in local mode)

emt> **xmit fileA**

emt displays the following message:

```
     Ready to transmit file fileA
```

Next, press **F1**. **emt** enters remote mode, and transmits fileA to the remote host.

If you type:

emt> **rcv fileB**

emt displays this message:

```
Ready to receive file fileB.
```

Next, enter remote mode by pressing **F1**. Use a remote host command to display the information that you want fileB to receive. **emt** automatically writes this and all subsequent host transmissions into fileB. To stop the **rcv**, press **F2**.

TRANSMISSION CONVENTIONS

Use the **emt** command **interm** to specify the line terminator used by the host. If you do not know what the host uses as a line terminator, experiment by changing **interm**. Use the **emt** command **outterm** to specify the line terminator to be transmitted to the host.

emt allows you to open only one Domain file at a time. If **emt** receives a **xmit** or **rcv** command while another Domain file is active, it closes the open Domain file, and executes the new command.

During remote mode, **emt** waits on both the keyboard and SIO line for characters to process, and monitors the data for characters of special interest to **emt**.

You can specify which keyboard characters **emt** should interpret by placing the keyboard in raw or cooked mode. In raw mode, **emt** passes all keyboard input (except the function keys, keys L1 through L12, and keys R1 through R4), directly to the host. Cooked mode lets you use many of the Display Manager's features for editing the input pad. **emt** places your keyboard in cooked mode by default.

COMMANDS

The following commands are available while running **emt**:

Command	Description
F1	Switch between local and remote modes.
F2	Interrupt a file transfer and close the file.
F3	Turn **tee** on or off. **tee** on causes **emt** to display file transmission records on the screen. You can use this feature to monitor file transfers, and decide if and when you should stop or interrupt a transfer. The default is **tee** on.
F8	Send a **break** to the host.
CTRL/F7	Display function key definitions.

These function keys may be simulated by typing the **emt** ESC character followed by the function key number (that is, ~1 for F1). When **emt** is used from the VT100 emulator, use shift **F1** instead of F2, and CTRL **F1** instead of F3.

Command	Description
ae	Abort on error.
asconly | notasconly	Sift out most non-printing ASCII codes. Eliminates triangles, allows BS, CR, ESC, FF, LF, TAB. The default is **notasc**.
break [*n*]	Set the **break** duration value to *n* milliseconds. The default is 200. If set to 0, the F8 (break) key does nothing.
close	Deactivate an **rcv** file. See the **rcv** command for related information.
code [*xx* | **none**]	Set the host-command-code to the hexadecimal number *xx*. The default is **none**.
cooked	Place the keyboard in cooked mode. This enables many DM features for editing the input pad, and provides an escape sequence for sending control characters to the remote host. To send the host a CTRL character, precede the character with a tilde (~). The sequence ~_ transmits a delete character. To send the host a single tilde character, type ~~.

The **emt** default is cooked mode. Cooked mode always echos keystrokes, so it does not require a full duplex connection to the host. (See the **raw** command for related information.)

Note: The **cooked** and **raw** commands refer only to the transcript pad and keyboard input. The SIO line itself is always in **raw** mode.

emtesc [chr|none]
Set the **emt** escape character to *chr*. Use none to disable the escape character. Default is ˜ for "cooked" mode, none for "raw" mode.

The following three commands are useful when standard input is redirected to a file of **emt** commands:

f1 Enter remote mode (Simulate function key F1).

f2 Terminate file transfer (Simulate function key f2).

f3 Toggle tee mode (Simulate function key F3).

hangup Cause modem to break connection with the remote host.

help [*tctl*] Display information about **emt** commands or about **tctl** commands.

line {1|2|3|*pathname*}
Select the SIO line. Pathname must specify an SIO device descriptor (for example, /dev/sio2). The default SIO line is 1 (/dev/sio1).

l Display the current SIO line, all **emt** switch settings and the receive filename, if any.

monit [*pathname*]
Write every character received over the SIO line to *pathname*. If a filename is not specified, the previous specification or error output is used.

nomonit Stop monitoring.

quit End the **emt** session.

raw [**–echo|–noecho**] [**–lf|–nolf**]
Place the keyboard in "raw" mode. This sends keyboard input directly to the remote host, interpreting only function keys. The **–echo** option echos keystrokes on standard output; you should use it when the host is in half-duplex mode. The default is **–noecho**. The **–lf** option converts carriage return (CR) to line feed (LF) for lines echoed. The default is **–nolf**. (See the **cooked** command for related information.) Note: The **–echo** and **–lf** options are purely local functions that enable you to read what you type. They do not in any way change host/node transmissions.

rcv [–**r**] [–**keys**|–**nokeys**] [*pathname*]
: Prepare the Domain file specified to receive remote host transmissions. If *pathname* already exists, **emt** appends the transmission to it, unless you specify –**r**. The receive begins when you enter remote mode **F1**. If you omit the *pathname*, **emt** uses the previous name, if any. The –**keys** option writes keystrokes to the file along with received data. The default is –**nokeys**.

  emt allows you to interrupt an **rcv** command at any time by pressing **F2**. **emt** remains in whatever mode it was in, but keeps the **rcv** file active. When you are ready to continue receiving host transmissions, you may type **rcv** again (in local mode) without a filename, and **emt** uses the same **rcv** file.

  If you omit filename and no **rcv** file is active, **emt** issues an error message. If you specify a new **rcv** file while another **rcv** file is active, **rcv** closes the active file, and prepares the new file to receive the transmission.

  Use the **close** command to deactivate an **rcv** file.

tctl {*tctl commands*}
: If you are running under Aegis, pass this command line to the shell command **tctl** to configure the SIO line. If this SIO line is not the default line, then you must use the –**line** command. The **speed** and **sync** commands have been superseded by this direct invocation of **tctl**. If only UNIX is installed, use **stty** to perform this action. If both UNIX and Aegis are installed, you can use either **tctl** or **stty**.

stty See **tctl**.

interm {**cr**|**lf**|**crlf**|**vax**|'*hex*'}
: Select the input line terminator. The default is **crlf**.

outterm {**cr**|**lf**|**crlf**|'*hex*'}
: Select the output line terminator. The default is **cr**. **emt** transmits the selected hexadecimal value as the terminator for each line.

xmit *pathname*
: Prepare to transmit the Domain file specified to the remote host. If you omit *pathname*, or if you specify a file that does not exist, **emt** issues an error message. When you issue this command, **emt** remains in local mode. **emt** transmits the file when you press **F1**.

When **emt** completes the transfer, it closes the file and returns to the previous mode. **emt** does not send an end-of-file (EOF) signal to the remote host. If the host requires an EOF, enter remote mode and transmit it manually.

emt can also receive commands from the host. If the host transmits the sequence

```
host-command-code (emt command string) line-terminator
```

emt interprets the string as an **emt** command. Use the **emt** command code to define [host-command-code].

Line Terminators	**emt Response**
crlf	Converts sequence to a line feed, ignoring any null characters that may separate the pair.
cr	Converts sequence to a line feed and ignores LFs.
lf	Interprets it as a line feed, and ignores CRs.
vax	Interprets both CR and CR-LF as terminators and converts them to line feed.
'hex'	Converts the given hexadecimal value to LF.

NAME

enable, disable – enable/disable LP printers

SYNOPSIS

enable *printers*
disable [**–c**] [**–r**[*reason*]] *printers*

DESCRIPTION

enable activates the named *printers*, enabling them to print requests taken by **lp**(1). Use **lpstat**(1) to find the status of printers.

disable deactivates the named *printers*, disabling them from printing requests taken by **lp**(1). By default, any requests that are currently printing on the designated printers are reprinted in their entirety either on the same printer or on another member of the same class. Use **lpstat**(1) to find the status of printers.

OPTIONS FOR DISABLE ONLY

–c	Cancels any requests that are currently printing on any of the designated printers.
–r[*reason*]	Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next **–r** option. If the **–r** option is not present or the **–r** option is given without a reason, a default reason is used. *Reason* is reported by **lpstat**(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).

NAME

env – set environment for command execution

SYNOPSIS

env [–] [name=value] ... [command args]

DESCRIPTION

env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The – flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

SEE ALSO

sh(1).
exec(2), profile(4), environ(5) in the *SysV Programmer's Reference*.

NAME

gdev: **hpd, erase, hardcopy, tekset, td** – graphical device routines and filters

SYNOPSIS

hpd [– *options*] [GPS *file* ...]
erase
hardcopy
tekset
td [–**ern***n*] [GPS *file* ...]

DESCRIPTION

All of the commands described below reside in **/usr/bin/graf** (see **graphics**(1G)).

hpd Translate a GPS (graphical primitive string; see gps(4)) to instructions for the Hewlett-Packard 7221A Graphics Plotter. A viewing window is computed from the maximum and minimum points in *file* unless the **–u** or **–r** *option* is provided. If no *file* is given, the standard input is assumed.

hpd Options

c*n*	Select character set *n*, *n* between 0 and 5.
p*n*	Select pen numbered *n*, *n* between 1 and 4 inclusive.
r*n*	Window on GPS region *n*, *n* between 1 and 25 inclusive.
s*n*	Slant characters *n* degrees clockwise from the vertical.
u	Window on the entire GPS universe.
xd*n*	Set x displacement of the viewport's lower left corner to *n* inches.
xv*n*	Set width of viewport to *n* inches.
yd*n*	Set y displacement of the viewport's lower left corner to *n* inches.
yv*n*	Set height of viewport to *n* inches.

erase Send characters to a Tektronix 4010 series storage terminal to erase the screen.

hardcopy When issued at a Tektronix display terminal with a hard copy unit, **hardcopy** generates a screen copy on the unit.

tekset Send characters to a Tektronix terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.

td Translate a GPS to scope code for a Tektronix 4010 series storage terminal. A viewing window is computed from the maximum and minimum points in *file* unless the **−u** or **−r** *option* is provided. If no *file* is given, the standard input is assumed.

td Options

e	Do not erase screen before initiating display.
r*n*	Display GPS region *n*, *n* between 1 and 25 inclusive.
u	Display the entire GPS universe.

SEE ALSO

graphics(1G).
gps(4) in the *SysV Programmer's Reference*.

NAME

esa – display address of external symbol

SYNOPSIS

esa *symbol_name*

DESCRIPTION

esa displays the address of an external symbol in an installed library. This command is primarily intended for system-level debugging.

symbol_name (required) Specify the symbol whose address you wish to display. esa is case sensitive with respect to the symbol name. Lowercase must be used to refer to symbols defined in FORTRAN and Pascal programs. Mixed case may be used, as needed, for symbols defined in C programs.

EXAMPLES

This command displays the address of **gpr_$init**. This symbol resides within the GPR library, which was installed at system start-up time.

```
$ esa gpr_$init
A1580C
$
```

NAME

ex – text editor

SYNOPSIS

ex [**–**] [**–v**] [**–t** *tag*] [**–r**] [**–R**] [*+command*] *name* ...

DESCRIPTION

ex is the root of a family of editors that includes **ex** and **vi**. **ex** is a superset of **ed**, with the most notable extension being a display editing capability. Display based editing is the focus of **vi**.

If you have a CRT terminal, you may wish to use a display based editor; in this case see **vi**(1), which is a command that focuses on the display editing portion of **ex**.

For ed Users

If you have used **ed** you will find that **ex** has a number of new features useful on CRT terminals. Intelligent terminals and high-speed terminals are very pleasant to use with **vi**. Generally, the editor uses far more of the capabilities of terminals than **ed** does, and uses the terminal capability data base and the type of the terminal you are using from the variable TERM in the environment to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using **vi**(1).

ex contains a number of new features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Pressing CTRL/D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just pressing return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

ex gives you more help when you make mistakes. **undo** (**u**) allows you to reverse any single change. **ex** gives you a lot of feedback, normally printing changed lines, and indicating when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also prevents overwriting existing files unless you edited them so that you do not accidentally clobber with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the telephone, you can use the editor **recover** command to retrieve your work. This gets you back to within a few lines of where you left off.

ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming file names and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

There is a command & in ex that repeats the last **substitute** command. In addition, there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. **ex** also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

INVOCATION OPTIONS

–	Suppresses all interactive-user feedback. Useful in processing editor scripts.
–v	Invokes **vi**.
–t *tagfl*	Edits the file containing the *tag* and positions the editor at its definition.
–r *file*	Recovers *file* after an editor or system crash. If *file* is not specified a list of all saved files is printed.
–R	*Readonly* mode set, prevents accidentally overwriting the file.
+*command*	Begins editing by executing the specified editor search or positioning *command*.

The *name* argument indicates files to be edited.

COMMAND NAMES AND ABBREVIATIONS

abbrev	**ab**	next	**n**	undo	**u**
append	**a**	number	**nu**	unmap	**unmap**
args	**ar**	preserve	**pre**	version	**ve**
change	**c**	print	**p**	visual	**vi**
copy	**co**	put	**pu**	write	**w**
delete	**d**	quit	**q**	xit	**x**
edit	**e**	read	**re**	yank	**ya**
file	**f**	recover	**rec**	window	**z**
global	**g**	rewind	**rew**	escape	**!**
insert	**i**	set	**se**	lshift	**<**
join	**j**	shell	**sh**	print next	**CR**
list	**l**	source	**so**	resubst	**&**
map		stop	**stop**	rshift	**>**
mark	**ma**	substitute	**s**	scroll	**^D**
move	**m**	unabbrev	**una**		

COMMAND ADDRESSES

n	line *n*	/*pat*	next with *pat*
.	current	?*pat*	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
−	previous	´*x*	marked with *x*
+*n*	*n* forward	´´	previous context
%	1,$		

STATES

Command	Normal and initial state. Input prompted for by :. Your kill character cancels partial command.
Insert	Entered by **a**, **i**, or **c**. Arbitrary text may be entered. Insert is normally terminated by a line having only . on it, or abnormally with an interrupt.
Visual	Entered by **vi**, terminates with **Q** or ^\

INITIALIZING OPTIONS

EXINIT	Place set's here in environment var.
$HOME/.exrc	Editor initialization file
./.exrc	Editor initialization file
set *x*	Enable option
set no*x*	Disable option
set *x=val*	Give value *val*
set	Show changed options
set all	Show all options
set *x*?	Show value of option *x*

MOST USEFUL OPTIONS

autoindent	ai	Supply indent
autowrite	aw	Write before changing files
ignorecase	ic	In scanning
list		Print ^I for tab, $ at end
magic		. [* special in patterns
number	nu	Number lines
paragraphs	para	Macro names which start ...
redraw		Simulate smart terminal
scroll		Command mode lines
sections	sect	Macro names ...
shiftwidth	sw	For < >, and input ^D
showmatch	sm	To) and } as typed
showmode	smd	Show insert mode in *vi*
slowopen	slow Stop updates during insert	
window		Visual mode lines
wrapscan	ws	Around end of buffer?
wrapmargin	wm	Automatic line splitting

SCANNING PATTERN FORMATION

^	Beginning of line
$	End of line
. Any character	
\<	Beginning of word
\>	End of word
[*str*]	Any char in *str*
[↑*str*]	... not in *str*
[*x*–*y*]	... between *x* and *y*
*	Any number of preceding

Vi ex

BUGS

The **undo** command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

undo never clears the buffer modified condition.

The z command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line '–' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

FILES

/usr/lib/ex?.?strings	Error messages
/usr/lib/ex?.?recover	Recover command
/usr/lib/ex?.?preserve	Preserve command
/usr/lib/*/*	Describes capabilities of terminals
$HOME/.exrc	Editor startup file
./.exrc	Editor startup file
/tmp/Ex*nnnnn*	Editor temporary
/tmp/Rx*nnnnn*	Named buffer temporary
/usr/preserve/*login*	Preservation directory (where *login* is the user's login)

SEE ALSO

awk(1), ed(1), edit(1), grep(1), sed(1), vi(1).
curses(3X), term(4), terminfo(4) in the *SysV Programmer's Reference*.

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2s complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

expr \| **expr**
: Returns the first **expr** if it is neither null nor **0**, otherwise returns the second **expr**.

expr \& **expr**
: Returns the first **expr** if neither **expr** is null or **0**, otherwise returns **0**.

expr { =, \>, \>=, \<, \<=, != } **expr**
: Returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, – } **expr**
: Addition or subtraction of integer-valued arguments.

expr { *, /, % } **expr**
: Multiplication, division, or remainder of the integer-valued arguments.

expr : **expr**
: The matching operator : compares the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of **ed**(1), except that all patterns are ''anchored'' (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(. . . \) pattern symbols can be used to return a portion of the first argument.

EXAMPLES

To add 1 to the shell variable **a**:

```
a=`expr $a + 1`
```

To return the last segment of a path name (i.e., file). Watch out for / alone as an argument: **expr** takes it as the division operator (see BUGS below).

```
'For $a equal to either "/usr/abc/file" or just "file"'
expr $a : '.*/\(.*\)' \| $a
```

To return the number of characters in $VAR:

```
expr $VAR : '.*'
```

BUGS

After argument processing by the shell, **expr** cannot tell the difference between an operator and an operand except by the value. If $a is an =, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to **expr** (and they are all be taken as the = operator). The following works:

```
expr X$a = X=
```

DIAGNOSTICS

As a side effect of expression evaluation, **expr** returns the following exit values:

0	If the expression is neither null nor **0**
1	If the expression *is* null or **0**
2	For invalid expressions.

syntax error	Operator/operand error.
non-numeric argument	Arithmetic was attempted on such a string.

SEE ALSO

ed(1), sh(1).

NAME

f77 – Fortran 77 compiler

SYNOPSIS

f77 [*options*] *files*

DESCRIPTION

f77 is the UNIX Fortran 77 compiler; it accepts several types of *file* arguments: Arguments whose names end with **.f** are taken to be Fortran 77 source programs; they are compiled, and each object program is left in the current directory in a file whose name is that of the source, with **.o** substituted for **.f**. Arguments whose names end with **.r** are taken to be RATFOR source programs. These are first transformed by the appropriate preprocessor, then compiled by **f77**, producing **.o** files. Arguments whose names end with **.c** are taken to be C source programs and are compiled, producing **.o** files. Arguments whose names end with **.e** or **.s** (EFL and assembly source programs) are not supported.

OPTIONS

The following *options* have the same meaning as in cc(1) (see **ld**(1) for link editor *options*):

–c	Suppresses link editing and produce **.o** files for each source file.
–O	Causes optimized code to be generated.
–o*output*	Names the final output file **output**, instead of **a.out**.
–g	Generates additional information needed for the use of **dbx**(1).

The following *options* are peculiar to **f77**:

–C Generates code for run-time subscript range checking.

–I[24] Changes the default size of integer variables (only valid on machines where the "normal" integer size is not equal to the size of a single precision real). **–I2** causes all integers to be 2-byte quantities. The default, **–I4**, causes all integers to be 4-byte quantities. (The **–Is** *option* is not supported.)

–v Prints the version number of the compiler, and the name of each pass as it executes.

–w Suppresses all warning messages. (**–w66** is not supported).

–F Applies the RATFOR preprocessor to relevant files, puts the result in files whose names have their suffix changed to **.f**. (No **.o** files are created.)

–m Applies the M4 preprocessor to each RATFOR source file before transforming it with the **ratfor**(1) processor.

–R The remaining characters in the argument are used as a RATFOR flag argument whenever processing a **.r** file.

The following *options* are not supported in the SysV version of **f77**: **–S**, **–f**, **–onetrip**, **–1**, **–66**, **–U**, **–u**, and **–E**.

Other arguments are taken to be either link-editor option arguments or **f77**-compilable object programs (typically produced by an earlier run), or libraries of **f77**-compilable routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the default name *a.out*.

FILES

file.[frc]	Input file
file.o	Object file
a.out	Linked output
/usr/lib/libF77.a	Intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/usr/apollo/lib/ftn	Compiler

The following files are not supported:

./fort[pid].?	Temporary
/usr/lib/f77pass1	Compiler
/usr/lib/f77pass2	Pass 2
/lib/c2	Optional optimizer

DIAGNOSTICS

The diagnostics produced by **f77** itself are intended to be self-explanatory. Occasional messages may be produced by the link editor, **ld**(1).

SEE ALSO

asa(1), cc(1), fsplit(1), ld(1), m4(1), prof(1), ratfor(1), dbx(1).

NAME

factor – obtain the prime factors of a number

SYNOPSIS

factor [integer]

DESCRIPTION

When you use **factor** without an argument, it waits for you to give it an integer. After you give it a positive integer less than or equal to 10^{14}, it factors the integer, prints its prime factors the proper number of times, and then waits for another integer. **factor** exits if it encounters a zero or any non-numeric character.

If you invoke **factor** with an argument, it factors the integer as described above, and then it exits.

The maximum time to factor an integer is proportional to $\sqrt{n}$. **factor** takes this time when n is prime or the square of a prime.

DIAGNOSTICS

Ouch	For input out of range or for garbage input.

NAME

true, **false** – provide truth values

SYNOPSIS

true

false

DESCRIPTION

true does nothing, successfully. **false** does nothing, unsuccessfully. They are typically used in input to **sh**(1) such as:

```
while true
do
        command
done
```

DIAGNOSTICS

true has exit status zero; **false** has exit status nonzero.

SEE ALSO

sh(1).

NAME

fgrep – search a file for a character string

SYNOPSIS

fgrep [*options*] *string* [*file* ...]

DESCRIPTION

fgrep (fast grep) seaches files for a character string and prints all lines that contain that string. **fgrep** is different from **grep**(1) and **egrep**(1) because it searches for a string, instead of searching for a pattern that matches an expression. It uses a fast and compact algorithm.

The characters $, *, [, ^, | , (,), and \ are interpreted literally by **fgrep**, that is, **fgrep** does not recognize *full regular expressions* like **egrep** does. Since these characters have special meaning to the shell, it is safest to enclose the entire *string* in single quotes '...'.

If no files are specified, **fgrep** assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

OPTIONS

–b	Precedes each line by the block number on which it was found. Useful in locating block numbers by context (first block is 0).
–c	Prints only a count of the lines that contain the pattern.
–i	Ignores upper/lower case distinction during comparisons.
–l	Prints the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.
–n	Precedes each line by its line number in the file (first line is 1).
–v	Prints all lines except those that contain the pattern.
–x	Prints only lines matched entirely.
–e *special_string*	Searches for a *special string* (*string* begins with a –).
–f *file*	Takes the list of *strings* from *file*.

BUGS

Ideally there should be only one **fgrep** command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**.

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

SEE ALSO

ed(1), egrep(1), grep(1), sed(1), sh(1).

NAME

file – determine file type

SYNOPSIS

file [**–c**] [**–f** *ffile*] [**–m** *mfile*] *arg* ...

DESCRIPTION

The **file** command performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, **file** examines the first 512 bytes and tries to guess its language. If an argument is an executable **a.out**, **file** prints the version stamp, if it is greater than zero.

The **file** command uses the file /etc/**magic** to identify files that have some sort of ''magic number'', that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of /etc/**magic** explains its format.

OPTIONS

–c	Check the magic file for format errors. This validation is not normally carried out, for efficiency reasons. No file typing is done under –c.
–f *ffile*	Take the next argument to be a file containing the names of the files to be examined.
–m *mfile*	Use an alternate magic file, *mfile*.

FILES

/etc/**magic**

SEE ALSO

filehdr(4)

NAME

find – find files

SYNOPSIS

find *path-name-list expression*

DESCRIPTION

find recursively descends the directory hierarchy for each pathname in the *path-name-list,* seeking files that match a Boolean *expression* written in the primaries given below. The SysV implementation of **find** does not follow symbolic links.

EXPRESSIONS

(In the descriptions below, the argument *n* is used as a decimal integer where +*n* means more than *n*, –*n* means less than *n*, and *n* means exactly *n*).

–name *file*	True if *file* matches the current filename. Normal shell argument syntax may be used if escaped, but watch out for [, ? and *.
–perm *onum*	True if the file permission flags exactly match the octal number *onum*. If *onum* is prefixed by a minus sign, only the bits that are set in *onum* are compared with the file permission flags, and the expression evaluates true if they match. Information about file permissions is found in **chmod**(1).
–type *c*	True if the type of the file is *c*, where *c* is *b* (block special file), *c* (character special file), *d* (directory), *p* (FIFO, or named pipe), *f* (plain file), or *l* (softlink).
–links *n*	True if the file has *n* links.
–user *uname*	True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a log-in name in the /etc/**passwd** file, it is taken as a user ID.
–group *gname*	True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the /etc/**group** file, it is taken as a group ID.
–size *n*[c]	True if the file is *n* blocks long (1024 bytes per block). If *n* is followed by a c, the size is in characters.
–atime *n*	True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by **find** itself.
–mtime *n*	True if the file has been modified in *n* days.
–ctime *n*	True if the file has been changed in *n* days.
–exec *cmd*	True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument {} is replaced by the current pathname.

–ok *cmd*	Like –exec, except this prints the generated command line with a question mark first, and executes only if you respond by typing y.
–print	Always true; print the current pathname.
–cpio *device*	Always true; write the current file on *device* in **cpio**(4) format (5120-byte records).
–newer *file*	True if the current file has been modified more recently than the argument *file*.
–depth	Always true; descend the directory hierarchy so that all entries in a directory are acted on before the directory itself. Can be useful when **find** is used with **cpio**(1) to transfer files contained in directories without write permission.
–mount	Always true; restricts the search to the file system containing the directory specified, or if no directory was specified, the current directory.
–local	True if the file physically resides on the local system. Note: This expression has no effect on Apollo systems.
(*expression*)	True if a parenthetical expression is true (parentheses are special to the shell and must be escaped).

OPERATORS

The primaries listed above may be combined using the following operators (in order of decreasing precedence):

1) The negation of a primary (! is the unary *not* operator).
2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
3) Alternation of primaries (–o is the *or* operator).

EXAMPLE

To remove all files named **a.out** or ***.o** that have not been accessed for a week:

```
# find / \( –name a.out –o –name '*.o' \) –atime +7 –exec rm {} \;
```

FILES

/etc/passwd
/etc/group

BUGS

find / /-depth always fails with the message:

```
find: stat failed: : No such file or directoiy
```

SEE ALSO

chmod (1), cpio (1), sh (1), test (1), stat (2), umask (2), cpio (4).

NAME

finger – user information lookup program

SYNOPSIS

finger [*options*] *name* ...

DESCRIPTION

By default **finger** lists the log-in name, full name, terminal name and write status (as a "*" before the terminal name if write permission is denied), idle time, log-in time, and office location and phone number (if they are known) for each current user. (Idle time is minutes if it is a single integer, hours and minutes if a ":" is present, or days and hours if a "d" is present.)

A longer format also exists and is used by **finger** whenever you specify a list of people's names. Account names as well as users' first and last names are accepted. This format is multiline, and includes all the information described above as well as the user's home directory and log-in shell, any plan which the person has placed in the file **.plan** in his home directory, and the project he is working on from the file **.project**, also in the home directory.

finger can be used to look up users on a remote machine. Specify the user as "*user@host*". If you omit the username, **finger** provides the standard format listing on the remote machine.

OPTIONS

–m	Match arguments only on username.
–l	Force long output format.
–p	Suppress printing of the **.plan** files
–s	Force short output format.

FILES

/etc/utmp	who file
/etc/passwd	For users names, offices, ...
/usr/adm/lastlog	Last log-in times
~/.plan	Plans
~/.project	Projects

NOTES

finger performs poorly in large registries, unless you use the **–m** option.

BUGS

finger prints only the first line of the **.project** file.

The encoding of the gcos field is UCB dependent.

You cannot pass arguments to the remote machine, as **finger** uses an internet standard port.

A user information database is in the works and will radically alter the way the information that **finger** uses is stored. **finger** will require extensive modification when this is implemented.

Domain/OS does not support **/usr/adm/lastlog**.

SEE ALSO

chfn(1), who(1)

NAME

french_to_iso – convert files to ISO format

SYNOPSIS

french_to_iso *input_file output_file*

DESCRIPTION

These utilities convert files written with the overloaded 7-bit national fonts to the Internation Standards Organization (ISO) 8-bit format. The overloaded fonts include any with a specific language suffix (for example, **f7x13.french**, or **din_f7x11.german**). If you created and/or edited a file using one of the national fonts, that file is a candidate for conversion.

You are not required to convert files, but probably will want to because

1. The overloaded fonts replace certain ASCII characters with national ones, have that subset of ASCII characters and the national characters in one file. The 8-bit fonts available as of SR10 include all the ASCII characters and the national characters.
2. The 8-bit fonts also include a wider range of national characters, so you can enter any character in any western European language. This was not always possible with the overloaded fonts. For example, there was not enough space in the overloaded font to include all the French characters, but they all exist in the 8-bit fonts.

There are two parameters to the conversion utilities. You must provide the name of the file you want to convert (*input_file*) and your *output_file*. If *output_file* already exists, the utilities abort.

The default location for the utilities is **/usr/apollo/bin**.

FILES

/usr/apollo/bin/french_to_iso	Converts overloaded French to ISO format
/usr/apollo/bin/german_to_iso	Converts overloaded German to ISO format
/usr/apollo/bin/nor.dan_to_iso	Converts overloaded Norwegian/Danish to ISO format
/usr/apollo/bin/swedish_to_iso	Converts overloaded Swedish/Finnish to ISO format
/usr/apollo/bin/swiss_to_iso	Converts overloaded Swiss to ISO format
/usr/apollo/bin/uk_to_iso	Converts overloaded U.K. English to ISO format

DIAGNOSTICS

All messages are generally self-explanatory.

NAME

fsplit – split FORTRAN or **ratfor files**

SYNOPSIS

fsplit *options files*

DESCRIPTION

The **fsplit** command splits the named *file(s)* into separate files, with one procedure per file. A procedure includes **blockdata**, **function**, **main**, **program**, and **subroutine** program segments. Normally, procedure X is put in file X.f or X.r, depending on the language option chosen. The following exceptions apply: **main** is put in the file **MAIN.[fr]**, and unnamed *blockdata* segments in the files **blockdata***N*.**[fr]** (where *N* is a unique integer value for each file).

OPTIONS

–f	Uses FORTRAN source program files as input.
–r	Uses **ratfor**(1) source program files as input.
–s	Strips FORTRAN input lines to 72 or fewer characters with trailing blanks removed.

SEE ALSO

csplit (1), ratfor (1), split (1).

NAME

fst – print fault status information

SYNOPSIS

fst [[**–s**] [**–r**] | [**–a**]] [**–u** *n*]

DESCRIPTION

fst prints information about the most recent fault that occurred in the process. The information always includes the fault status, the program counter (PC) at which the fault occurred, and a textual description of the error as reported by the system call **error_$print**. **fst** is intended for system-level debugging.

If you are using a Peripheral Bus Unit (PBU) device, you can get fault information by using the **–u** option (see below).

fst is obsolete and is valid only when running in INPROCESS compatibility mode with the **inprocess** variable set and all commands running in-process. Use the command tb **–full instead of fst.**

OPTIONS

–r Print the contents of the CPU general registers when the fault occurred.

–s Print the supervisor PC, entry control block (ECB), and status register (SR) if the fault occurred in supervisor mode. This option is ignored if the fault occurred in user mode.

–a Print all available fault information. (Prints the same information as both **–s** and **–r**.)

–u *n* Print the same information as both **–s** and **–r** for faults caused by the PBU interrupt handler for unit *n*.

EXAMPLES

$ **fst –a**

```
Fault Diagnostic Information
Fault Status   = 00120010:
process quit (from OS / fault handler)
User Fault PC  = 000157FC
D0-D7: 00120010 00000000 00000002 FFFFFFFE 00000008 00000006 \
00000182 00000004
A0-A7: 0020A452 00E2F22E 0020A3D4 0020A450 00E2F174 0000C92C \
002746B4 002746AC
Supervisor ECB = 00000000
Supervisor SR  = 0000
Supervisor PC  = 00000000
```

NAME

ftp – ARPANET file transfer program

SYNOPSIS

ftp [–v] [–d] [–i] [–n] [–g] [host]

DESCRIPTION

ftp is the user interface to the ARPANET standard File Transfer Protocol (FTP). The program allows you to transfer files to and from a remote network site.

You can specify the client host with which **ftp** is to communicate on the command line. If you do, **ftp** immediately attempts to establish a connection to an FTP server on that host; otherwise, **ftp** enters its command interpreter and awaits instructions from you. When **ftp** is awaiting commands from you, it displays the prompt ''ftp>''.

OPTIONS

You can specify options on the command line, or to the command interpreter.

–v (verbose on) Forces **ftp** to show all responses from the remote server, as well as report on data transfer statistics.

–n Restrains **ftp** from attempting ''auto-login'' on initial connection. If auto-login is enabled, **ftp** checks the **.netrc** (see below) file in your home directory for an entry describing an account on the remote machine. If no entry exists, **ftp** prompts for the remote machine log-in name (the default is the user identity on the local machine), and, if necessary, prompts for a password and an account with which to log in.

–i Turns off interactive prompting during multiple file transfers.

–d Enables debugging.

–g Disables filename globbing.

COMMANDS

! [*command* [*args*]]
Invoke an interactive shell on the local machine. If you specify arguments, **ftp** takes the first to be a command to execute directly, with the rest of the arguments as its arguments.

$ *macro-name* [*args*]
Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro unglobbed.

account [*passwd*]
Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If you do not specify an argument, **ftp** prompts you for an account password in a non-echoing input mode.

append *local-file* [*remote-file*]
Append a local file to a file on the remote machine. If you do not specify

remote-file, **ftp** uses the local filename, after applying the changes required by any **ntrans** or **nmap** setting, to name the remote file. **ftp** uses the current settings for **type**, **form**, **mode**, and **structure**.

ascii Set the file transfer type to network ASCII. This is the default type.

bell Arrange that a bell be sounded after each file transfer command is completed.

binary Set the file transfer type to support binary image transfer.

bye Terminate the FTP session with the remote server and exit **ftp**. An end-of-file also terminates the session and exits.

case Toggle remote computer filename case-mapping during **mget** commands. When **case** is on (the default is off), remote computer filenames with all letters in uppercase are written in the local directory with the letters mapped to lowercase.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

cdup Change the remote-machine working directory to the parent of the current remote-machine working directory.

close Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

cr Toggle carriage-return stripping during ASCII-type file retrieval. Records are denoted by a carriage-return/linefeed sequence during ASCII-type file transfer. When **cr** is on (the default), carriage returns are stripped from this sequence to conform with the UNIX single-linefeed record delimiter. Records on non-UNIX remote systems may contain single linefeeds; when an ASCII-type transfer is made, you can distinguish these linefeeds from a record delimiter only when **cr** is off.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If you specify an optional *debug-value*, **ftp** uses it to set the debugging level. When debugging is on, **ftp** prints each command sent to the remote machine, preceded by the string "-->".

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, place the output in *local-file*. If you do not specify a directory, **ftp** uses the current working directory on the remote machine. If you do not specify a local file, or *local-file* is –, **ftp** sends output to the terminal.

disconnect A synonym for **close**.

form *format* Set the file transfer form to *format*. The default and only supported format is **file**.

get *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If you do nt specify the local filename, **ftp** gives it the same name it has on the remote machine, subject to alteration by the current **case**, **ntrans**, and **nmap** settings. **ftp** uses the current settings for **type**, **form**, **mode**, and **structure** while transferring the file.

glob Toggle filename expansion for **mdelete**, **mget** and **mput**. If you turn globbing off with **glob**, **ftp** takes the filename arguments literally and does not expand them. Globbing for **mput** is done as in **csh**(1). For **mdelete** and **mget**, each remote filename is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name is likely to be different from expansion of an ordinary filename: the exact result depends on the foreign operating system and FTP server, You can preview the results by executing '**mls** *remote-files* –'. Note: **mget** and **mput** are not meant to transfer entire directory subtrees of files. You can do that by transferring a **tar**(1) archive of the subtree (in binary mode).

hash Toggle hash-sign (#) printing for each data block transferred. The size of a data block is 1024 bytes.

help [*command*]
Print an informative message about the meaning of *command*. If you do not specify an argument, **ftp** prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If you do not specify a *directory*, **ftp** uses your home directory.

ls [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. If you do not specify *remote-directory* , **ftp** uses the current working directory. If you do not specify a local file, or if *local-file* is –, **ftp** sends the output to the terminal.

macdef *macro-name*
Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until you execute a **close** command. The macro processor interprets '$' and '\' as special characters. A '$' followed by a number (or numbers) is replaced by the corresponding argument on the macro-

invocation command line. A '$' followed by an 'i' signals that macro processor that the executing macro is to be looped. On the first pass '$i' is replaced by the first argument on the macro-invocation command line, on the second pass it is replaced by the second argument, and so on. A '\' followed by any character is replaced by that character. Use the '\' to prevent special treatment of the '$'.

mdelete [*remote-files*]
Delete the *remote-files* on the remote machine.

mdir *remote-files local-file*
This command works like **dir**, except that you can specify multiple remote files. If interactive prompting is on, **ftp** prompts you to verify that the last argument is indeed the target local file for receiving **mdir** output.

mget *remote-files*
Expand the *remote-files* on the remote machine and execute a **get** for each filename thus produced. See **glob** for details on the filename expansion. Resulting filenames are then processed according to **case**, **ntrans**, and **nmap** settings. Files are transferred into the local working directory, which you can change with '**lcd** *directory*'; You can create new local directories with '**! mkdir** *directory*'.

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
This command is like **ls**, except that you can specify multiple remote files. If interactive prompting is on, **ftp** prompts you to verify that the last argument is indeed the target local file for receiving **mls** output.

mode [*mode-name*]
Set the file transfer **mode** to *mode-name*. The default and only supported *mode-name* is **stream**.

mput *local-files*
Expand wildcards in the list of local files given as arguments and execute a **put** for each file in the resulting list. See **glob** for details of filename expansion. Resulting filenames are then processed according to **ntrans** and **nmap** settings.

nmap [*inpattern outpattern*]
Set or unset the filename-mapping mechanism. If you do not specify an argument, the filename-mapping mechanism is unset. If you specify an argument, **nmap** maps remote filenames during **mput** commands and **put** commands issued without a specified remote-target filename. and maps local filenames during **mget** commands and **get** commands issued without a specified local-target filename. This command is useful when

you are connecting to a non-UNIX remote computer with different file-naming conventions or practices.

The mapping follows the pattern set by *inpattern* and *outpattern*. *Inpattern* is a template for incoming filenames (which may have already been processed according to the **ntrans** and case settings). Include the sequences '$1', '$2', ..., '$9' in *inpattern*, if you want variable templating. Use '\' to prevent this special treatment of the '$' character. **nmap** treats all other characters literally, and uses them to determine the **nmap** *inpattern* variable values. For example, given *inpattern* $1.$2 and the remote filename "**mydata.data**", $1 has the value "**mydata**", and $2 has the value "**data**".

The *outpattern* determines the resulting mapped filename. The sequences '$1', '$2', ..., '$9' are replaced by any value resulting from the *inpattern* template. The sequence '$0' is replaced by the original filename. Additionally, the sequence '[*seq1*,*seq2*]' is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*. For example, the command **nmap $1.$2.$3 [$1,$2].[$2,file]** yields the output filename **myfile.data** for input filenames **myfile.data** and **myfile.data.old**, **myfile.file** for the input filename **myfile**; and **myfile.myfile** for the input filename **.myfile**. You can include spaces in *outpattern*, as in the example: **nmap $1 |sed "s/ *$//"** > **$1** . Use the '\' character to prevent special treatment of the '$', '[', ']', and ',' characters.

ntrans [*inchars* [*outchars*]]

Set or unset the filename-character-translation mechanism. If you do not specify an argument, the filename-character-translation mechanism is unset. If you specify an argument, **ntrans** translates characters in remote filenames during **mput** commands and **put** commands issued without a specified remote-target filename, and translates characters in local filenames during **mget** commands and **get** commands issued without a specified local-target filename.

This command is useful when you are connecting to a non-UNIX remote computer with different file-naming conventions or practices. **ntrans** replaces characters in a filename matching a character in *inchars* with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, **ntrans** deletes the character from the filename.

open *host* [*port*]

Establish a connection to the specified *host* FTP server. You can specify an optional port number, in which case **ftp** attempts to contact an FTP server at that port. If the **auto-login** option is on (default), **ftp** also attempts to automatically log you in to the FTP server (see below).

prompt Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow you to selectively retrieve or store files. If prompting is turned off (default is on), any **mget** or **mput** transfers all files, and any **mdelete** deletes all files.

proxy *ftp-command*
Execute an **ftp** command on a secondary control connection. This command allows you to connect simultaneously to two remote FTP servers for transferring files between them. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command **proxy ?** to see other **ftp** commands executable on the secondary connection. The following commands behave differently when prefaced by **proxy**: **open** does not define new macros during the auto-login process, **close** does not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mput**, and **append** transfer files from the host on the secondary control connection to the host on the primary control connection. Third-party file transfers depend upon support of the FTP protocol PASV command by the server on the secondary control connection.

put *local-file* [*remote-file*]
Store a local file on the remote machine. If you do not specify *remote-file*, **put** uses the local filename after processing according to any **ntrans** or **nmap** settings in naming the remote file. **ftp** uses the current settings for **type**, **form**, **mode**, and **structure**.

pwd Print the name of the current working directory on the remote machine.

quit This is a synonym for **bye**.

quote *arg1 arg2 ...*
This command sends the arguments you specify, verbatim, to the remote FTP server.

recv *remote-file* [*local-file*]
This is a synonym for **get**.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

rename [*from*] [*to*]
Rename the file *from* on the remote machine, to the file *to*.

reset Clear the reply queue. This command resynchronizes command/reply sequencing with the remote FTP server. Resynchronization may be necessary following a violation of the FTP protocol by the remote server.

rmdir *directory-name*
Delete the specified directory on the remote machine.

runique Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a **get** or **mget** command, **runique** appends a **.1** to the name. If the resulting name matches another existing file, **runique** appends a .2 to the original name. If this process continues up to **.99**, **runique** prints an error message. **ftp** does not execute the transfer, and reports the generated unique filename. Note that **runique** does not affect local files generated from a shell command (see below). The default value is off.

send *local-file* [*remote-file*]
This is a synonym for **put**.

sendport Toggle the use of PORT commands. By default, **ftp** attempts to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when you perform multiple file transfers. If the PORT command fails, **ftp** uses the default data port. When the use of PORT commands is disabled,**ftp** does not attempt to use PORT commands for each data transfer. This is useful for certain FTP implementations that ignore PORT commands but indicate, incorrectly, that they are accepted.

status Show the current status of **ftp**.

struct [*struct-name*]
Set the file transfer structure to *struct-name*; either **stream** or **record**. By default, **struct** uses **stream** structure.

sunique Toggle storing of files on remote machine under unique filenames. The remote FTP server must support the FTP protocol STOU command for successful completion. The remote server reports unique names. The default value is off.

tenex Set the file transfer type to that needed to talk to TENEX machines.

trace Toggle packet tracing.

type [*type-name*]
Set the file transfer type to *type-name*; one of **ascii**, **binary**, **image**, or **tenex**. If you do not specify a type, **type** prints the current type. The default type is **ascii** (network ASCII).

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, **ftp** will prompt the user for it (after disabling local echo). If the FTP server requires an account field and you do not specify it, **ftp** prompts for it. If you specify an account field, **ftp** relays an account command to the remote server after the log-in

sequence is completed, if the remote server did not require it for logging in. Unless you invoke **ftp** with "auto-login" disabled, **ftp** executes this process automatically, on initial connection to the FTP server.

verbose Toggle verbose mode. In verbose mode, **ftp** displays all responses from the FTP server and also reports statistics regarding the efficiency of a file transfer, when the transfer completes. By default, verbose is on.

? [*command*]
This is a synonym for **help**.

You can enclose command arguments that have embedded spaces in quotation (") marks.

ABORTING A FILE TRANSFER

Use the terminal interrupt key (usually CTRL/C) to abort a file transfer. **ftp** immediately stops sending transfers. You can stop receiving transfers by sending a FTP protocol ABOR command to the remote server and discarding any further data received. The speed at which this is accomplished depends on the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an "ftp>" prompt does not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence is ignored when **ftp** has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above, or from unexpected behavior by the remote server, including violations of the FTP protocol. If the delay results from unexpected remote server behavior, you must kill the local **ftp** program by hand.

FILE-NAMING CONVENTIONS

ftp processes files that you specify as arguments according to the following rules:

1) If you specify the filename as a dash (–), **ftp** uses **stdin** (for reading) or **stdout** (for writing).

2) If the first character of the filename is "|", **ftp** interprets the remainder of the argument as a shell command, then forks a shell, using **popen**(3) with the argument you specify, and reads (writes) from **stdout** (**stdin**). If the shell command includes spaces, you must enclose the argument in quotation marks; for example, ""| ls –lt"". A particularly useful example of this mechanism is "**dir |more**".

3) Failing the above checks, if globbing is enabled, **ftp** expands local filenames according to the rules used in the **csh**(1); see the **glob** command for a comparison. If **ftp** expects a single local file (for example, **put**), it uses only the first filename generated by the "globbing" operation.

4) For **mget** commands and get commands with unspecified local filenames, the local filename is the remote filename, that a **case**, **ntrans**, or **nmap** setting can change. The remote server can then change the resulting filename, if **runique** is on.

5) For **mput** commands and **put** commands with unspecified remote filenames, the remote filename is the local filename, that a **ntrans** or **nmap** setting can change. he remote server can then change the resulting filename, if **sunique** is on.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters that may affect a file transfer. The type can be one of **ascii**, **image** (binary), **ebcdic**, and **local byte size**. **ftp** supports the **ascii** and **image** types of file transfer, plus **local byte size 8** for **tenex** mode transfers.

ftp supports only the default values for the remaining file transfer parameters: **mode**, **form**, and **struct**.

THE .netrc FILE

The **.netrc** file contains log-in and initialization information used by the auto-login process. It resides in your home directory. **.netrc** recognizes the following tokens; you can separate them by spaces, tabs, or newlines:

machine *name*	Identify a remote machine name. The auto-login process searches the **.netrc** file for a **machine** token that matches the remote machine specified on the **ftp** command line or as an **open** command argument. Once a match is made, the subsequent **.netrc** tokens are processed, stopping when the end-of-file is reached or another **machine** token is encountered.
login *name*	Identify a user on the remote machine. If this token is present, the auto-login process initiates a login using the specified *name*.
password *string*	Supply a password. If this token is present, the auto-login process supplies the *string* if the remote server requires a password as part of the log-in process. Note that if this token is present in the **.netrc** file, **ftp** aborts the auto-login process if the **.netrc** is readable by anyone besides the user.
account *string*	Supply an additional account password. If this token is present, the auto-login process supplies the *string* if the remote server requires an additional account password, or the auto-login process initiates an ACCT command if it does not.

macdef *name*	Define a macro. This token functions like the **ftp macdef** command functions. A macro is defined with the specified *name*; its contents begin with the next **.netrc** line and continue until a null line (consecutive new-line characters) is encountered. If a macro named **init** is defined, **ftp** automatically executes it as the last step in the auto-login process.

BUGS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2BSD UNIX ASCII-mode transfer code has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2BSD servers using the ASCII type. Avoid this problem by using the binary image type.

NAME

gdev: **hpd, erase, hardcopy, tekset, td** – graphical device routines and filters

SYNOPSIS

hpd [– *options*] [GPS *file* ...]
erase
hardcopy
tekset
td [–ern*n*] [GPS *file* ...]

DESCRIPTION

All of the commands described below reside in **/usr/bin/graf** (see **graphics**(1G)).

hpd Translate a GPS (graphical primitive string; see **gps**(4)) to instructions for the Hewlett-Packard 7221A Graphics Plotter. A viewing window is computed from the maximum and minimum points in *file* unless the **–u** or **–r** *option* is provided. If no *file* is given, the standard input is assumed.

hpd Options

c*n*	Select character set *n*, *n* between 0 and 5.
p*n*	Select pen numbered *n*, *n* between 1 and 4 inclusive.
r*n*	Window on GPS region *n*, *n* between 1 and 25 inclusive.
s*n*	Slant characters *n* degrees clockwise from the vertical.
u	Window on the entire GPS universe.
xd*n*	Set x displacement of the viewport's lower left corner to *n* inches.
xv*n*	Set width of viewport to *n* inches.
yd*n*	Set y displacement of the viewport's lower left corner to *n* inches.
yv*n*	Set height of viewport to *n* inches.

erase Send characters to a Tektronix 4010 series storage terminal to erase the screen.

hardcopy When issued at a Tektronix display terminal with a hard copy unit, **hardcopy** generates a screen copy on the unit.

tekset Send characters to a Tektronix terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.

td Translate a GPS to scope code for a Tektronix 4010 series storage terminal. A viewing window is computed from the maximum and minimum points in *file* unless the **−u** or −r *option* is provided. If no *file* is given, the standard input is assumed.

td Options

e	Do not erase screen before initiating display.
r*n*	Display GPS region *n*, *n* between 1 and 25 inclusive.
u	Display the entire GPS universe.

SEE ALSO

graphics(1G).
gps(4) in the *SysV Programmer's Reference*.

NAME

german_to_iso – convert files to ISO format

SYNOPSIS

german_to_iso *input_file output_file*

DESCRIPTION

These utilities convert files written with the overloaded 7-bit national fonts to the Internation Standards Organization (ISO) 8-bit format. The overloaded fonts include any with a specific language suffix (for example, **f7x13.french**, or **din_f7x11.german**). If you created and/or edited a file using one of the national fonts, that file is a candidate for conversion.

You are not required to convert files, but probably will want to because

1. The overloaded fonts replace certain ASCII characters with national ones, have that subset of ASCII characters and the national characters in one file. The 8-bit fonts available as of SR10 include all the ASCII characters and the national characters.
2. The 8-bit fonts also include a wider range of national characters, so you can enter any character in any western European language. This was not always possible with the overloaded fonts. For example, there was not enough space in the overloaded font to include all the French characters, but they all exist in the 8-bit fonts.

There are two parameters to the conversion utilities. You must provide the name of the file you want to convert (*input_file*) and your *output_file*. If *output_file* already exists, the utilities abort.

The default location for the utilities is **/usr/apollo/bin**.

FILES

/usr/apollo/bin/french_to_iso	Converts overloaded French to ISO format
/usr/apollo/bin/german_to_iso	Converts overloaded German to ISO format
/usr/apollo/bin/nor.dan_to_iso	Converts overloaded Norwegian/Danish to ISO format
/usr/apollo/bin/swedish_to_iso	Converts overloaded Swedish/Finnish to ISO format
/usr/apollo/bin/swiss_to_iso	Converts overloaded Swiss to ISO format
/usr/apollo/bin/uk_to_iso	Converts overloaded U.K. English to ISO format

DIAGNOSTICS

All messages are generally self-explanatory.

NAME

get – get a version of an SCCS file

SYNOPSIS

get [**–r**SID] [**–c***cutoff*] [**–i***list*] [**–x***list*] [**–w***string*] [**–a***seq-no.*] [**–k**] [**–e**] [**–l**[**p**]] [**–p**] [**–m**] [**–n**] [**–s**] [**–b**] [**–g**] [**–t**] *file* . . .

DESCRIPTION

get generates an ASCII text file from each named SCCS file according to the specifications given by its *options*, which begin with –. You can specify *options* in any order, but all *options* apply to all named SCCS files. If a directory is named, **get** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the **g–file** whose name is derived from the SCCS filename by simply removing the leading s.; (see also *FILES*, below).

Each of the *options* is explained below as though only one SCCS file is to be processed, but the effects of any *option* applies independently to each named file.

OPTIONS

–r*SID* The CCS IDentification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by **delta**(1) if the –e *option* is also used), as a function of the SID specified.

–c*cutoff* *Cutoff* date-time, in the form *YY*[*MM*[*DD*[*HH*[*MM*[*SS*]]]]] No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **–c7502** is equivalent to **–c750228235959**. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date-time. This feature allows you to specify a *cutoff* date in the form: "**–c77/2/2 9:22:25**". Note this implies that you can use the %E% and %U% identification keywords (see below) for nested **gets** within, say the input to a **send**(1C) command:

```
~!get "–c%E% %U%" s.file
```

–i*list* Forces a *list* of deltas to be included in the creation of the generated file.

The *list* has the following syntax:

```
<list> ::= <range> | <list>, <range>
<range> ::= SID | SID – SID
```

SID, the SCCS Identification of a delta, can be in any form shown in the "SID Specified" column of Table 1.

–x*list*	Forces a *list* of deltas to be excluded in the creation of the generated file. See the **–i** *option* for the *list* format.
–e	Indicates the **get** is for the purpose of editing or making a change (delta) to the SCCS file with the subsequent use of **delta**(1). The –e *option* used in a get for a particular version (SID) of the SCCS file prevents further gets from editing on the same SID until the delta is executed or the **j** (joint edit) flag is set in the SCCS file (see **admin**(1)). Concurrent use of **get** –e for different SIDs is always allowed. If the **g–file** generated by **get** with an –e *option* is accidentally ruined in the process of editing it, you can regenerate it by re-executing the **get** command with the **–k** *option* in place of the –e *option.* SCCS file protection specified by the ceiling, floor, and authorized user list stored in the SCCS file (see **admin**(1)) are enforced when the –e *option* is used.
–b	Used with the –e *option* to indicate that the new delta should have an SID in a new branch as shown in Table 1. This *option* is ignored if the **b** flag is not present in the file (see **admin**(1)) or if the retrieved delta is not a leaf delta. (A leaf delta is one that has no successors on the SCCS file tree.) Note: You can always create a branch delta from a non-leaf delta. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.
–k	Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **–k** *option* is implied by the –e *option.*
–l[p]	Causes a delta summary to be written into an **l–file**. If **–lp** is used then an **l–file** is not created; the delta summary is written on the standard output instead. See **FILES** for the format of the **l–file**.
–p	Causes the text retrieved from the SCCS file to be written on the standard output. No **g–file** is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the –s *option* is used, in which case it disappears.
–s	Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

−m	Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.
−n	Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **−m** and **−n** *options* are used, the format is: %M% value, followed by a horizontal tab, followed by the **−m** *option* generated format.
−g	Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an **l−file**, or to verify the existence of a particular SID.
−t	Accesses the most recently created delta in a given release (e.g., **−r1**), or release and level (e.g., **−r1.2**).
−w *string*	Substitutes *string* for all occurrences of %W% when getting the file.
−a*seq-no.*	The delta sequence number of the SCCS file delta (version) to be retrieved (see sccsfile(5)). This *option* is used by the **comb**(1) command; it is not a generally useful *option*. If both the **−r** and −a *options* are specified, only −a is used. Take care when using −a in conjunction with −e, as the SID of the delta to be created may not be what you expect. **−r** can be used with −a and −e to control the naming of the SID of the delta to be created.

For each file processed, get responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If −e is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each filename is printed (preceded by a new-line) before it is processed. If −i is used included deltas are listed following the notation ''Included''; if −x is used, excluded deltas are listed following the notation ''Excluded''.

TABLE 1. Determination of SCCS Identification String

SID* Specified	−b Option Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	–	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	–	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch succ.	R.L.B.S	R.L.(mB+1).1

* ‘‘R’’, ‘‘L’’, ‘‘B’’, and ‘‘S’’ are the ‘‘release’’, ‘‘level’’, ‘‘branch’’, and ‘‘sequence’’ components of the SID, respectively; ‘‘m’’ means ‘‘maximum’’. Thus, for example, ‘‘R.mL’’ means ‘‘the maximum level number within release R’’; ‘‘R.L.(mB+1).1’’ means ‘‘the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R’’. Note that if the SID specified is of the form ‘‘R.L’’, ‘‘R.L.B’’, or ‘‘R.L.B.S’’, each of the specified components *must* exist.

** ‘‘hR’’ is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

# Successor.

† The **−b** *option* is effective only if the **b** flag (see **admin**(1)) is present in the file. An entry of – means ‘‘irrelevant’’.

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	**Value**
%M%	Module name: either the value of the **m** flag in the file (see **admin**(1)), or if absent, the name of the SCCS file with the leading s. removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file (see **admin**(1)).
%F%	SCCS filename.
%P%	Fully qualified SCCS filename.
%Q%	The value of the q flag in the file (see **admin**(1)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string @(#) recognizable by **what**(1).
%W%	A shorthand notation for constructing **what**(1) strings for UNIX system program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing **what**(1) strings for non-UNIX system program files. %A% = %Z%%Y% %M% %I%%Z%

Several auxiliary files may be created by **get**. These files are known generically as the **g–file**, **l–file**, **p–file**, and **z–file** . The letter before the hyphen is called the tag. An auxiliary filename is formed from the SCCS filename: the last component of all SCCS filenames must be of the form s.*module-name*, the auxiliary files are named by replacing the leading s with the tag. The **g–file** is an exception to this scheme: g–**file** is named by removing the s. prefix. For example, s.xyz.c, the auxiliary filenames would be xyz.c, **l**.xyz.c, **p**.xyz.c, and z.xyz.c, respectively.

The **g–file**, which contains the generated text, is created in the current directory (unless the f3–p *option* is used). A **g–file** is created in all cases, whether or not any lines of text were generated by the **get**. It is owned by the real user. If **–k** is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The **l–file** contains a table showing which deltas were applied in generating the retrieved text. The **l–file** is created in the current directory if the **–l** *option* is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the **l**-file have the following format:

- a. A blank character if the delta was applied;
  * otherwise.
- b. A blank character if the delta was applied or was not applied and ignored;
  * if the delta was not applied and was not ignored.
- c. A code indicating a ''special'' reason why the delta was or was not applied:
  ''I'': Included.
  ''X'': Excluded.
  ''C'': Cut off (by a **–c** keyletter).
- d. Blank.
- e. SCCS identification (SID).
- f. Tab character.
- g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
- h. Blank.
- i. Login name of person who created **delta**.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The **p–file** is used to pass information resulting from a **get** with an **–e** *option* along to **delta**. Its contents are also used to prevent a subsequent execution of get with an **–e** *option* for the same SID until **delta** is executed or the joint edit flag, **j**, (see **admin**(1)) is set in the SCCS file. The **p–file** is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the **p–file** is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the **get** was executed, followed by a blank and the **–i** argument if it was present, followed by a blank and the –x argument if it was present, followed by a newline. There can be an arbitrary number of lines in the **p**-file at any time; no two lines can have the same new delta SID.

The **z–file** serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., **get**) that created it. The **z–file** is created in the directory containing the SCCS file for the duration of **get**. The same protection restrictions as those for the **p–file** apply for the **z–file**. The **z–file** is created mode 444.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the **–e** *option* is used.

FILES

g–file	Existed before the execution of **delta**; removed after completion of **delta**.
p–file	Existed before the execution of **delta**; may exist after completion of **delta**.
q–file	Created during the execution of **delta**; removed after completion of **delta**.
x–file	Created during the execution of **delta**; renamed to SCCS file after completion of **delta**.
z–file	Created during the execution of **delta**; removed during the execution of **delta**.
d–file	Created during the execution of **delta**; removed after completion of **delta**.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the **g–file**.

DIAGNOSTICS

Use **help**(1) for explanations.

SEE ALSO

admin(1), delta(1), help(1), prs(1), sccs(1), what(1).

NAME

getopt – parse command options

SYNOPSIS

**set — `getopt** *optstring* $*`

DESCRIPTION

getopt breaks up options in command lines for easy parsing by shell procedures and checks for legal options. *optstring* is a string of recognized option letters (see **getopt**(3C)); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space.

The special option — is used to delimit the end of the options. If it is used explicitly, **getopt** recognizes it; otherwise, **getopt** generates it; in either case, **getopt** places it at the end of the options.

The positional parameters ($1 $2 ...) of the shell are reset so that each option is preceded by a – and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

You should begin using the new command **getopts**(1) in place of **getopt**. **getopt** will not be supported in the next major release. For more information, see the **WARNINGS** section.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set — `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b)       FLAG=$i; shift;;
        -o)     OARG=$2; shift 2;;
        —)      shift; break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

WARNINGS

getopt does not support the part of Rule 8 of the command syntax standard (see **intro**(1)) that permits groups of option-arguments following an option to be separated by white space and quoted. For example,

```
cmd -a -b -o "xxx z yy" file
```

is not handled correctly). To correct this deficiency, use the new command **getopts** in place of **getopt**.

getopt will not be supported in the next major release. For this release a conversion tool has been provided, **getoptcvt**(1). For more information about **getopts** and **getoptcvt**, see the **getopts** manual page.

If an option that takes an option-argument is followed by a value that is the same as one of the options listed in *optstring*, referring to the earlier EXAMPLE section, but using the following command line:

```
cmd -o -a file
```

getopt always treats **–a** as an option-argument to **–o**; it never recognizes **–a** as an option. In this case, the **for** loop in the example shifts past the *file* argument.

DIAGNOSTICS

getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

SEE ALSO

getopts(1), getoptcvt(1), sh(1).
getopt(3C) in the *SysV Programmer's Reference*.

NAME

getopts, getoptcvt – parse command options

SYNOPSIS

getopts *optstring name* [*arg* ...]"

/usr/lib/getoptcvt [–b] *file*

DESCRIPTION

getopts parses positional parameters for shell procedures and checks for legal options. It supports all applicable rules of the command syntax standard (see Rules 3-10, **intro**(1)). It should be used in place of the **getopt**(1) command. (See the **WARNING-** section.)

optstring must contain the option letters recognized by the command using **getopts**; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, **getopts** placea the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell procedure is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, **getopts** places it in the shell variable **OPTARG**.

If an illegal option is encountered, ? is placed in *name*.

When **getopts** reaches the end of options, it exits with a non-zero exit status. The special option "—" can be used to delimit the end of the options.

By default, **getopts** parses positional parameters. If extra arguments (*arg* ...) are given on the **getopts** command line, **getopts** parses them instead.

/usr/lib/getoptcvt reads the shell script in *file*, converts it to use **getopts** instead of **getopt**, and writes the results on the standard output.

So all new commands adhere to the command syntax standard described in **intro**, they should use **getopts** or **getopt**(3C) to parse positional parameters and check for legal options (see the **WARNING** section.)

OPTIONS

–b Results of running **/usr/lib/getoptcvt** are portable to earlier releases of the UNIX system. **/usr/lib/getoptcvt** modifies the shell script in *file* so that when the resulting shell script is executed, it determines at run time whether to invoke **getopts** or **getopt**.

EXAMPLE

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
        case $c in
        a | b)       FLAG=$c;;
        o)           OARG=$OPTARG;;
        \?)          echo $USAGE
                     exit 2;;
        esac
done
shift `expr $OPTIND - 1`
```

This code will accept any of the following as equivalent:

```
cmd -a -b -o "xxx z yy" file
cmd -a -b -o "xxx z yy" -- file
cmd -ab -o xxx,z,yy file
cmd -ab -o "xxx z yy" file
cmd -o xxx,z,yy -b -a file
```

WARNING

Although the following command syntax rule (see **intro**) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLE** section, **a** and **b** are options, and the option **o** requires an option-argument:

```
cmd -aboxxx file
```

Rule 5 violation: options with option-arguments must not be grouped with other options.

```
cmd -ab -oxxx file
```

Rule 6 violation: there must be white space after an option that takes an option-argument.

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

DIAGNOSTICS

getopts prints an error message on the standard error when it encounters an option letter not included in *optstring*.

SEE ALSO

intro(1), sh(1).
getopts(3C) in the *SysV Programmer's Reference*.

NAME

getopts, getoptcvt – parse command options

SYNOPSIS

getopts optstring name [arg ...]

/usr/lib/getoptcvt [**–b**] file

DESCRIPTION

getopts parses positional parameters for shell procedures and checks for legal options. It supports all applicable rules of the command syntax standard (see Rules 3-10, **intro**(1)). It should be used in place of the **getopt**(1) command. (See the **WARNING**- section.)

optstring must contain the option letters recognized by the command using **getopts**; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, **getopts** placea the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell procedure is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, **getopts** places it in the shell variable **OPTARG**.

If an illegal option is encountered, ? is placed in *name*.

When **getopts** reaches the end of options, it exits with a non-zero exit status. The special option "—" can be used to delimit the end of the options.

By default, **getopts** parses positional parameters. If extra arguments (*arg* ...) are given on the **getopts** command line, **getopts** parses them instead.

/usr/lib/getoptcvt reads the shell script in *file*, converts it to use **getopts** instead of **getopt**, and writes the results on the standard output.

So all new commands adhere to the command syntax standard described in **intro**, they should use **getopts** or **getopt**(3C) to parse positional parameters and check for legal options (see the **WARNING**section.)

OPTIONS

–b Results of running **/usr/lib/getoptcvt** are portable to earlier releases of the UNIX system. **/usr/lib/getoptcvt** modifies the shell script in *file* so that when the resulting shell script is executed, it determines at run time whether to invoke **getopts**(1) or **getopt**(1).

EXAMPLE

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
      case $c in
      a | b)    FLAG=$c;;
      o)        OARG=$OPTARG;;
      \?)       echo $USAGE
                exit 2;;
      esac
done
shift `expr $OPTIND - 1`
```

This code will accept any of the following as equivalent:

```
cmd -a -b -o "xxx z yy" file
cmd -a -b -o "xxx z yy" -- file
cmd -ab -o xxx,z,yy file
cmd -ab -o "xxx z yy" file
cmd -o xxx,z,yy -b -a file
```

WARNING

Although the following command syntax rule (see **intro**) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLE** section above, **a** and **b** are options, and the option **o** requires an option-argument:

```
cmd -aboxxx file
```

Rule 5 violation: options with option-arguments must not be grouped with other options.

```
cmd -ab -oxxx file
```

Rule 6 violation: there must be white space after an option that takes an option-argument.

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

DIAGNOSTICS

getopts prints an error message on the standard error when it encounters an option letter not included in *optstring*.

SEE ALSO

intro(1), sh(1).
getopts(3C) in the *SysV Programmer's Reference*.

NAME

graph – draw a graph

SYNOPSIS

graph [*options*]

DESCRIPTION

graph with no *options* takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the **tplot**(1G) filters.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels can be surrounded with quotes ", in which case they may be empty or contain blanks and numbers; labels never contain newlines.

A legend indicating grid range is produced with a grid unless –s is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

OPTIONS

The following options are recognized, each as a separate argument:

–a	Supplies abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by –x).
–b	Breaks (disconnects) the graph after each label in the input.
–c	Character string given by next argument is default label for each point.
–g	Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).
–l	Next argument is label for graph.
–m	Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (e.g., the Tektronix 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).
–s	Saves screen, does not erase before plotting.
–x [l]	If l is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) x limits. Third argument, if present, is grid spacing on x axis. Normally these quantities are determined automatically.
–y [l]	Similarly for y.
–h	Next argument is fraction of space for height.
–w	Similarly for width.
–r	Next argument is fraction of space to move right before plotting.

−u	Similarly to move up before plotting.
−t	Transposes horizontal and vertical axes. (−x now applies to the vertical axis.)

BUGS

graph stores all points internally and drops those for which there is no room.

Segments that run out of bounds are dropped, not windowed.

Logarithmic axes may not be reversed.

SEE ALSO

graphics(1G), spline(1G), tplot(1G).

NAME

graphics – access graphic and numeric commands

SYNOPSIS

graphics [–r]

DESCRIPTION

graphics prefixes the path name **/usr/bin/graf** to the current **$PATH** value, changes the primary shell prompt to ^, and executes a new shell. The directory **/usr/bin/graf** contains all of the **graphics** subsystem commands.

The format for a **graphics** command is *command name argument*(s). An *argument* can be a *filename* or an *option string*. A *filename* is the name of any UNIX system file except those beginning with –. The *filename* – is the name for the standard input. An *option string* consists of – followed by one or more *option*(s). An *option* consists of a keyletter possibly followed by a value. *Options* may be separated by commas.

The graphic commands consist of: Commands that manipulate and plot numeric data (see **stat**(1G)). Commands that generate tables of contents (see **toc**(1G)). Commands that interact with graphic devices (see **gdev**(1G)) and **ged**(1G)). A collection of graphic utility commands (see **gutil**(1G)).

You can generate a list of the **graphics** commands by typing **whatis** in the **graphics** environment.

OPTION

–r Creates a restricted environment for access to the graphical commands. That is, **$PATH** is set to:

```
:/usr/bin/graf:/rbin:/usr/rbin:/bin:/usr/bin
```

and the restricted shell, **rsh**, is invoked. To restore the environment that existed prior to issuing the **graphics** command, type **EOT** (CTRL/D on most terminals). To logoff from the **graphics** environment, type **quit**.

SEE ALSO

gdev(1G), gutil(1G), stat(1G), toc(1G).
gps(4) in the *SysV Programmer's Reference*.

NAME

greek – select terminal filter

SYNOPSIS

greek [–T*terminal*]

DESCRIPTION

greek is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character Teletype Model 37 terminal for certain other terminals. Special characters are simulated by overstriking, if necessary and possible. If the argument is omitted, **greek** attempts to use the environment variable **$TERM** (see **environ**(5)). Currently, the following *terminal*s are recognized:

300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	Diablo 1620 (alias DASI 450).
1620-12	Diablo 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	Tektronix 4014.
hp	Hewlett-Packard 2621, 2640, and 2645.
tek	Tektronix 4014.

FILES

/usr/bin/300
/usr/bin/300s
/usr/bin/4014
/usr/bin/450
/usr/bin/hp

SEE ALSO

300(1), 4014(1), 450(1), hp(1), eqn(1), mm(1), nroff(1). tplot(1G).
environ(5), greek(5), term(5) in the *SysV Programmer's Reference*.

NAME

grep – search a file for a pattern

SYNOPSIS

grep [*options*] *limited regular expression* [*file* ...]

DESCRIPTION

grep searches files for a pattern and prints all lines that contain that pattern. **grep** uses *limited regular expressions* (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with **ed**(1) to match the patterns. It uses a compact non-deterministic algorithm.

Be careful using the characters $, *, [, ^, | , (,), and \ in the *limited regular expression* because they are also meaningful to the shell. It is safest to enclose the entire *limited regular expression* in single quotes '...'.

If no files are specified, **grep** assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

OPTIONS

–b	Precedes each line by the block number on which it was found. Useful in locating block numbers by context (first block is 0).
–c	Prints only a count of the lines that contain the pattern.
–i	Ignores upper/lower case distinction during comparisons.
–l	Prints the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.
–n	Precedes each line by its line number in the file (first line is 1).
–s	Suppresses error messages about nonexistent or unreadable files.
–v	Prints all lines except those that contain the pattern.

BUGS

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**.

If there is a line with embedded nulls, **grep** only matches up to the first null; if that matches, it prints the entire line.

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

SEE ALSO

ed(1), egrep(1), fgrep(1), sed(1), sh(1).

NAME

gutil – graphic utilities

SYNOPSIS

command-name [*options*] [*files*]

DESCRIPTION

Below is a list of miscellaneous device-independent utility commands found in **/usr/bin/graf**. If no *files* are given, input is from the standard input. All output is to the standard output. Graphic data is stored in GPS format; see **gps**(4).

bel Sends bel character to terminal.

cvrtopt [=s*string* f*string* i*string* t*string*] [*args*]

Converts *options*. Reformats *args* (usually the command line arguments of a calling shell procedure) to facilitate processing by shell procedures. An *arg* is either a file name (a string not beginning with a –, or a – by itself) or an *option* string (a string of *options* beginning with a –). Output is of the form:

–option –option . . . file name(s)

All *options* appear singularly and preceding any filenames. *Options* that take values (e.g., –r1.1) or are two letters long must be described through *options* to **cvrtopt**.

cvrtopt is usually used with set in the following manner as the first line of a shell procedure:

set – `**cvrtopt** =[*options*] $@`

Options to **cvrtopt** are:

s*string*	*String* accepts string values.
f*string*	*String* accepts floating point numbers as values.
i*string*	*String* accepts integers as values.
t*string*	*String* is a two-letter *option* name that takes no value.

String is a one- or two-letter *option* name.

gd [GPS*files*] – GPS dump

Prints a human-readable listing of GPS.

gtop [−r*n*u] [GPS*files*]

GPS to **plot**(4) filter. Transforms a GPS into **plot**(4) commands displayable by **plot** filters. GPS objects are translated if they fall within the window that circumscribes the first *file* unless an *option* is given. *Options* to **gtop** are:

r*n*	Translates objects in GPS region *n*.
u	Translates all objects in the GPS universe.

pd [**plot**(5)*files*]

plot(4) dump. Prints a human-readable listing of **plot**(4) format graphic commands.

ptog [**plot**(5)*files*]

plot(4) to GPS filter. Transforms **plo**(4) commands into a GPS.

quit Terminates session.

remcom [*files*]

Remove comments. Copies its input to its output with comments removed. Comments are as defined in C (i.e., /* comment */).

whatis [−o] [*names*]

Brief online documentation. Prints a brief description of each *name* given. If no *name* is given, then the current list of description *names* is printed. The command **whatis*** prints out every description. Using the −o option causes only command *options* to be printed.

yoo*file* Pipe fitting primitive. Deposits the output of a pipeline into a *file* used in the pipeline. Without yoo, this is not usually successful as it causes a read and write on the same file simultaneously.

SEE ALSO

graphics(1G).
gps(4), plot(4) in the *SysV Programmer's Reference*.

NAME

gdev: **hpd**, **erase**, **hardcopy**, **tekset**, **td** – graphical device routines and filters

SYNOPSIS

hpd [– *options*] [GPS *file* ...]
erase
hardcopy
tekset
td [–**ern***n*] [GPS *file* ...]

DESCRIPTION

All of the commands described below reside in **/usr/bin/graf** (see **graphics**(1G)).

hpd Translate a GPS (graphical primitive string; see **gps**(4)) to instructions for the Hewlett-Packard 7221A Graphics Plotter. A viewing window is computed from the maximum and minimum points in *file* unless the **–u** or **–r** *option* is provided. If no *file* is given, the standard input is assumed.

hpd Options

c*n*	Select character set *n*, *n* between 0 and 5.
p*n*	Select pen numbered *n*, *n* between 1 and 4 inclusive.
r*n*	Window on GPS region *n*, *n* between 1 and 25 inclusive.
s*n*	Slant characters *n* degrees clockwise from the vertical.
u	Window on the entire GPS universe.
xd*n*	Set x displacement of the viewport's lower left corner to *n* inches.
xv*n*	Set width of viewport to *n* inches.
yd*n*	Set y displacement of the viewport's lower left corner to *n* inches.
yv*n*	Set height of viewport to *n* inches.

erase Send characters to a Tektronix 4010 series storage terminal to erase the screen.

hardcopy When issued at a Tektronix display terminal with a hard copy unit, **hardcopy** generates a screen copy on the unit.

tekset Send characters to a Tektronix terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.

td Translate a GPS to scope code for a Tektronix 4010 series storage terminal. A viewing window is computed from the maximum and minimum points in *file* unless the **−u** or **−r** *option* is provided. If no *file* is given, the standard input is assumed.

td Options

e	Do not erase screen before initiating display.
r*n*	Display GPS region *n*, *n* between 1 and 25 inclusive.
u	Display the entire GPS universe.

SEE ALSO

graphics(1G).
gps(4) in the *SysV Programmer's Reference*.

NAME

spell, hashmake, spellin, hashcheck – find spelling errors

SYNOPSIS

spell [**–v**] [**–b**] [**–x**] [**–l**] [*+local_file*] [*files*]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

spell ignores most **troff**(1), **tbl**(1), and **eqn**(1) constructions.

By default, **spell** follows chains of included files (**.so** and **.nx** **troff**(1) requests), *unless* the names of such included files begin with **/usr/lib**. Under the **–l** option, **spell** will follow the chains of *all* included files.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy–y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by **spell**:

hashmake	Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.
spellin	Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.
hashcheck	Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

OPTIONS

The following options apply to **spell**:

–v	Prints all words not literally in the spelling list, and indicate plausible derivations from the words in the spelling list.
–b	Checks British spelling. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

−x	Prints every plausible stem with = for each word.
+*local_file*	Removes words found in *local_file* are removed from **spell**'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to **spell**'s own spelling list) for each job.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]
Hashed spelling lists, American & British

S_SPELL=/usr/lib/spell/hstop
Hashed stop list

H_SPELL=/usr/lib/spell/spellhist
History file

/usr/lib/spell/spellprog
Program

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.

SEE ALSO

sed(1), sort(1), tee(1).

NAME

spell, hashmake, spellin, hashcheck – find spelling errors

SYNOPSIS

spell [–v] [**–b**] [–x] [**–l**] [*+local_file*] [*files*]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

spell ignores most **troff**(1), **tbl**(1), and **eqn**(1) constructions.

By default, **spell** follows chains of included files (**.so** and **.nx troff**(1) requests), *unless* the names of such included files begin with **/usr/lib**. Under the **–l** option, **spell** will follow the chains of *all* included files.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy–y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by **spell**:

hashmake	Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.
spellin	Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.
hashcheck	Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

OPTIONS

The following options apply to **spell**:

–v	Prints all words not literally in the spelling list, and indicate plausible derivations from the words in the spelling list.
–b	Checks British spelling. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

−x	Prints every plausible stem with = for each word.
+*local_file*	Removes words found in *local_file* are removed from spell's output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to spell's own spelling list) for each job.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]
Hashed spelling lists, American & British

S_SPELL=/usr/lib/spell/hstop
Hashed stop list

H_SPELL=/usr/lib/spell/spellhist
History file

/usr/lib/spell/spellprog
Program

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.

SEE ALSO

sed(1), sort(1), tee(1).

NAME

help – ask for SCCS help

SYNOPSIS

help [*arguments*]

DESCRIPTION

help finds information to explain the use of, or a particular message from, an SCCS command.

Arguments can be either message numbers (which normally appear in parentheses following messages), command names, or one of the following types:

type 1	Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines that produced the message. For example, ge3 means message 3 from the get(1) command.
type 2	Does not contain numerics. That is, it appears as a command name such as the name get.
type 3	Is all numeric. For example, 26.

FILES

/usr/lib/help	Directory containing files of message text
/usr/lib/help/helploc	File containing locations of help files not in **/usr/lib/help**

NAME

hostid – set or print identifier of current host system

SYNOPSIS

hostid [*identifier*]

DESCRIPTION

The **hostid** command prints the *identifier* of the current host in hexadecimal. This numeric value is expected to be unique across all hosts and is commonly set to the host's Internet address. The super-user can set the host ID by giving a hexadecimal argument or the hostname; this is usually done in the start-up script **/etc/rc.local**.

NAME

hostname – set or print name of current host system

SYNOPSIS

hostname [*name-of-host*]

DESCRIPTION

The **hostname** command prints the name of the current host. The super-user can set the host name by giving an argument; this is usually done in the start-up script **/etc/rc.local**.

NAME

hp – handle special functions of Hewlett-Packard terminals

SYNOPSIS

hp [–e] [–m]

DESCRIPTION

hp supports special functions of the Hewlett-Packard 2640 series of terminals, with the primary purpose of producing accurate representations of most **nroff** output. A typical usage is as follows:

nroff –h *files* ... **hp**

Regardless of the hardware options on your terminal, **hp** tries to do sensible things with underlining and reverse linefeeds. If the terminal has the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it has the "mathematical-symbol" feature, Greek and other special characters can be displayed.

OPTIONS

–e Assumes that your terminal has the "display enhancements" feature, and so makes maximum use of the added display modes. Overstruck characters are shown underlined. Superscripts are shown half-bright, and subscripts in half-bright, underlined mode. If this *option* is omitted, **hp** assumes that your terminal lacks the "display enhancements" feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in inverse video.

–m Requests minimization of output by removal of newlines. Any contiguous sequence of 3 or more newlines is converted into a sequence of only 2 newlines; i.e., any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other special characters, **hp** provides the same set as does **300**(1), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown.

NOTES

The exit codes are **0** for normal termination, **2** for all errors.

BUGS

An "overstriking sequence" is defined as a printing character, followed by a backspace, followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in inverse video; otherwise, only the first printing character is shown (again, underlined or in Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (e.g., reverse linefeeds, backspaces) can make text "disappear"; in particular, tables generated by **tbl**(1) that contain vertical lines will often be missing the lines of text that contain the "foot" of a vertical line, unless the input to **hp** is piped through **col**(1).

Although some terminals do provide numerical superscript characters, no attempt is made to display them.

DIAGNOSTICS

line too long	The representation of a line exceeds 1,024 characters.

SEE ALSO

300(1), col(1), eqn(1), greek(1). nroff(1), tbl(1).

NAME

hpc – program counter histogram

SYNOPSIS

hpc [**–low** *x*] [**–high** *x*]
[**–from** *procedure*]
[**–to** *procedure*]
[**–proc** *procedure*]
[**–limit** *n*] [**–rate** *n*]
[**–nhdr**] [**–map**]
[**–brief**] *pathname*
[*args...*]

DESCRIPTION

hpc (histogram_program_counter), part of **Domain/PAK (Domain Performance** Analysis Kit), looks at the performance of programs at the PC level.

hpc produces a histogram of the program counter (PC) during program execution, thus helping you locate the most compute-bound portions of your program.

While your program is executing, **hpc** samples the PC at regular intervals, gathering a set of data points. Each data point records the region in which the program was executing the location of the PC when the sample was taken.

hpc divides your program into 256 equally sized regions called "buckets." The size of the region depends on the size of your program or the range you select. The smaller the region, the better the resolution of the analysis.

When execution of your program has ended, **hpc** displays statistics and a histogram (bar graph) of the PC. Each bar corresponds to an area of program memory. The length of the bar indicates how much time the program spent executing in the corresponding area. **hpc** tells you which procedures and line numbers each bar represents.

While **hpc** and your program are executing, a serial line is not available for output.

pathname (required)	Specify the name of the program to be evaluated.
args (optional)	Specify any arguments to be passed to the program *pathname*. These are not processed by **hpc**, but passed directly to your program. Default if omitted: no arguments passed

OPTIONS

If no options are specified, a histogram is produced for the entire program, with 500 samples taken per second.

–**low** *x*	Specify lowest address *x* to be included in the histogram. *x* must be a hexadecimal value. If this option is omitted, the histogram starts at the beginning of the program or procedure (see –**from** below).
–**high** *x*	Specify highest address *x* to be included in the histogram. *x* must be a hexadecimal value. If this option is omitted, the histogram continues to the end of the program or procedure (see –**to** below).
–**from** *procedure*	Specify the beginning of a procedure as the lowest address to be included in the histogram. If both –**from** and –**low** are omitted, the histogram starts at the beginning of the program. Note the the procedure name is case-insensitive.
–**to** *procedure*	Specify the end of a procedure as the highest address to be included in the histogram. If both –**to** and –**high** are omitted, the histogram stops at the end of the program. Note the the procedure name is case-insensitive.
–**proc** *procedure*	Specify a single procedure to be included in the histogram. Note the the procedure name is case-insensitive.

By limiting the range of addresses in the histogram with –**low**, –**high**, –**from**, –**to**, and –**proc**, you can study a specific part of your program, such as an I/O routine.

–**limit** *n*	Limit the displayed histogram bars to those that represent more than *n*% of the monitored program execution. The default value for *n* is 1. Use –**limit** 0 to show all histogram entries.
–**rate** *n*	Specify how many times *n* **hpc** samples the program counter per second. *n* must be a decimal number in the range 5 to 2000. The default is 500 samples per second. A higher rate results in a more accurate histogram, but tends to slow program execution.
–**nhdr**	Generate the histogram without the header information. Using this option makes filtering the output easier.
–**map**	Generate a list of the names and starting and ending locations of the procedures in the program. This list is reduced if –**from**, –**to**, –**high**, or –**low** are used to restrict monitoring to specific procedures or memory addresses. The output from this option can be quite verbose for large programs.

–brief Produce a compact bar chart by showing only the name of the first procedure, or procedure fragment, contained in the bucket represented by each bar. By default, **dpat** shows the names of all procedures or procedure fragments contained in the bucket. This option also suppresses source-line information.

SEE ALSO

dpat(1), dspst(1)

NAME

gdev: **hpd**, **erase**, **hardcopy**, **tekset**, **td** – graphical device routines and filters

SYNOPSIS

hpd [– *options*] [GPS *file* ...]
erase
hardcopy
tekset
td [–**ern***n*] [GPS *file* ...]

DESCRIPTION

All of the commands described below reside in **/usr/bin/graf** (see **graphics**(1G)).

hpd Translate a GPS (graphical primitive string; see **gps**(4)) to instructions for the Hewlett-Packard 7221A Graphics Plotter. A viewing window is computed from the maximum and minimum points in *file* unless the –**u** or –**r** *option* is provided. If no *file* is given, the standard input is assumed.

hpd Options

c*n*	Select character set *n*, *n* between 0 and 5.
p*n*	Select pen numbered *n*, *n* between 1 and 4 inclusive.
r*n*	Window on GPS region *n*, *n* between 1 and 25 inclusive.
s*n*	Slant characters *n* degrees clockwise from the vertical.
u	Window on the entire GPS universe.
xd*n*	Set x displacement of the viewport's lower left corner to *n* inches.
xv*n*	Set width of viewport to *n* inches.
yd*n*	Set y displacement of the viewport's lower left corner to *n* inches.
yv*n*	Set height of viewport to *n* inches.

erase Send characters to a Tektronix 4010 series storage terminal to erase the screen.

hardcopy When issued at a Tektronix display terminal with a hard copy unit, **hardcopy** generates a screen copy on the unit.

tekset Send characters to a Tektronix terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.

td Translate a GPS to scope code for a Tektronix 4010 series storage terminal. A viewing window is computed from the maximum and minimum points in *file* unless the −**u** or −r *option* is provided. If no *file* is given, the standard input is assumed.

td Options

e	Do not erase screen before initiating display.
r*n*	Display GPS region *n*, *n* between 1 and 25 inclusive.
u	Display the entire GPS universe.

SEE ALSO

graphics(1G).
gps(4) in the *SysV Programmer's Reference*.

NAME

id – print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

id writes a message on the standard output, giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

SEE ALSO

logname (1), getuid (2).

NAME

inlib – install a user-supplied library

SYNOPSIS

inlib *pathname...*

DESCRIPTION

inlib installs a library at the current shell level; it remains installed until the shell that installed it exits. See the note below for information on loading a library that is used by all processes. The newly installed library will be used to resolve external references of programs (and libraries) loaded after its installation. (Thus, previously loaded libraries and programs will not be affected.)

Note that only those global references that are marked by the binder become visible, and that the default action of the binder is to leave globals unmarked. Therefore, you should take care to mark all appropriate globals when you bind your library.

inlib is an internal shell command. You can create a library that is installed automatically in every process. This library resides in the file **/lib/userlib.private**. The procedure text in this library will be shared among all processes.

This library must be present at node start-up time in order to be installed. After copying your library to **/lib/userlib.private**, you must shut down the node and start it up again in order to use the library. Changes to the library also require rebooting the node to load the new routines.

Global names in **/lib/userlib.private** must not duplicate names used in Domain libraries.

pathname (required)	Specify name of library file(s) to be installed. Multiple pathnames and wildcarding are permitted.

NAME

intm – install a type manager

SYNOPSIS

intm [*options*] *type_name* [*mgr_pathname*]

DESCRIPTION

intm installs a type manager for the *type_name*. The manager is copied into the type manager directory from *mgr_pathname*. If *mgr_pathname* is omitted, the file named *type_name* in the current directory is used. The **intm** command does not accept wildcards.

type_name (required) Specify the type for which the manager is to be installed.

mgr_pathname (optional)
Specify the pathname of the manager object file to install for this type.

Default if omitted: object file is named *type_name*

OPTIONS

–n *node_spec*	Specify the node on which the type manager is to be installed. If this option is omitted, the type manager is installed on the current node.
–l	List the results of the operation.
–r	Replace an existing type manager if it exists.

EXAMPLES

```
$ intm example_type /mydir/my_example_mgr.bin
```

```
$ intm example_type /mydir/old_example_mgr.bin –n //remote_vol –l
"/mydir/old_example_mgr.bin" installed as the manager for
type example_type on volume //remote_vol.
```

SEE ALSO

inty(1)

NAME

inty – install a new type

SYNOPSIS

inty *[options]* type_name source_volume [**–n** *node_spec*]

DESCRIPTION

inty installs a type from one node to another. It installs both the type name and type manager on the target node (given by the **–n** option).

type_name (required) Specify the name of the type to be installed.

source_volume (required)
Specify the pathname of the source volume from which to copy the type name and type manager.

OPTIONS

–n *node_spec*	Specify the node on which the type is to be installed. You may also specify the entry directory of a volume mounted for software installation, as shown in the example below. If this option is omitted, the type is installed on the current node.
–l	List the results of the installation.
–r	Replace any existing type name/manager pair.

EXAMPLES

```
$ inty example_type //test_vol
Type "example_type" installed.


$ inty example_type //my_vol -n //test_vol -l
Type "example_type" installed on volume //test_vol.
```

SEE ALSO

crty(1), dlty(1),lty(1), intm(1)

NAME

ipcrm – remove a message queue, semaphore set, or shared memory id

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

ipcrm removes one or more specified messages, a semaphore, or shared memory identifiers.

OPTIONS

–q *msqid*	Removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.
–m *shmid*	Removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
–s *semid*	Removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.
–Q *msgkey*	Removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.
–M *shmkey*	Removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
–S *semkey*	Removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

Details of the removes are described in **msgctl**(2), **shmctl**(2), and **semctl**(2). Find identifiers and keys by using **ipcs**(1).

SEE ALSO

ipcs(1).
msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2) in the *SysV Programmer's Reference*.

NAME

ipcs – report inter-process communication facilities status

SYNOPSIS

ipcs [*options*]

DESCRIPTION

ipcs prints certain information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system.

OPTIONS

–q	Prints information about active message queues.
–m	Prints information about active shared memory segments.
–s	Prints information about active semaphores.

If **–q**, **–m**, or **–s** are specified, information about only those indicated is printed. If none of these three are specified, information about all three is printed subject to these options:

–b	Prints biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.
–c	Prints creator's login name and group name.
–o	Prints information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)
–p	Prints process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments.)
–t	Prints time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, last **semop**(2) on semaphores.)
–a	Uses all print *options*. (This is a shorthand notation for **–b**, **–c**, **–o**, **–p**, and **–t**.)

The column headings and the meaning of the columns in an **ipcs** list are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities are listed.

COLUMNS

T	(all)	Type of the facility:
		q message queue;
		m shared memory segment;
		s semaphore.

ID	(all)	The identifier for the facility entry.
KEY	(all)	The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)
MODE	(all)	The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows: The first two characters are: **R** if a process is waiting on a *msgrcv*; **S** if a process is waiting on a *msgsnd*; **D** if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it; **C** if the associated shared memory segment is to be cleared when the first attach is executed; **–** if the corresponding special flag is not set. The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused. The permissions are indicated as follows: **r** if read permission is granted; **w** if write permission is granted; **a** if alter permission is granted; **–** if the indicated permission is *not* granted.
OWNER	(all)	The login name of the owner of the facility entry.
GROUP	(all)	The group name of the group of the owner of the facility entry.
CREATOR	(a,c)	The login name of the creator of the facility entry.
CGROUP	(a,c)	The group name of the group of the creator of the facility entry.

CBYTES	(a,o)	The number of bytes in messages currently outstanding on the associated message queue.
QNUM	(a,o)	The number of messages currently outstanding on the associated message queue.
QBYTES	(a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue.
LSPID	(a,p)	The process ID of the last process to send a message to the associated queue.
LRPID	(a,p)	The process ID of the last process to receive a message from the associated queue.
STIME	(a,t)	The time the last message was sent to the associated queue.
RTIME	(a,t)	The time the last message was received from the associated queue.
CTIME	(a,t)	The time when the associated entry was created or changed.
NATTCH	(a,o)	The number of processes attached to the associated shared memory segment.
SEGSZ	(a,b)	The size of the associated shared memory segment.
CPID	(a,p)	The process ID of the creator of the shared memory entry.
LPID	(a,p)	The process ID of the last process to attach or detach the shared memory segment.
ATIME	(a,t)	The time the last attach was completed to the associated shared memory segment.
DTIME	(a,t)	The time the last detach was completed on the associated shared memory segment.
NSEMS	(a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME	(a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/unix	System namelist
/etc/passwd	User names
/etc/group	Group names

SEE ALSO

msgop(2), semop(2), shmop(2) in the *SysV Programmer's Reference*.

NAME

iso – convert files to ISO format

SYNOPSIS

french_to_iso *input_file output_file*
german_to_iso *input_file output_file*
nor.dan_to_iso *input_file output_file*
swedish_to_iso *input_file output_file*
swiss_to_iso *input_file output_file*
uk_to_iso *input_file output_file*

DESCRIPTION

These utilities convert files written with the overloaded 7-bit national fonts to the Internation Standards Organization (ISO) 8-bit format. The overloaded fonts include any with a specific language suffix (for example, **f7x13.french**, or **din_f7x11.german**). If you created and/or edited a file using one of the national fonts, that file is a candidate for conversion.

You are not required to convert files, but probably will want to because

1. The overloaded fonts replace certain ASCII characters with national ones, have that subset of ASCII characters and the national characters in one file. The 8-bit fonts available as of SR10 include all the ASCII characters and the national characters.
2. The 8-bit fonts also include a wider range of national characters, so you can enter any character in any western European language. This was not always possible with the overloaded fonts. For example, there was not enough space in the overloaded font to include all the French characters, but they all exist in the 8-bit fonts.

There are two parameters to the conversion utilities. You must provide the name of the file you want to convert (*input_file*) and your *output_file*. If *output_file* already exists, the utilities abort.

The default location for the utilities is **/usr/apollo/bin**.

FILES

/usr/apollo/bin/french_to_iso	Converts overloaded French to ISO format
/usr/apollo/bin/german_to_iso	Converts overloaded German to ISO format
/usr/apollo/bin/nor.dan_to_iso	Converts overloaded Norwegian/Danish to ISO format
/usr/apollo/bin/swedish_to_iso	Converts overloaded Swedish/Finnish to ISO format
/usr/apollo/bin/swiss_to_iso	Converts overloaded Swiss to ISO format
/usr/apollo/bin/uk_to_iso	Converts overloaded U.K. English to ISO format

DIAGNOSTICS

All messages are generally self-explanatory.

NAME

join – relational database operator

SYNOPSIS

join [*options*] *file1 file2*

DESCRIPTION

join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is –, the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line [see **sort**(1)].

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the *options* below use the argument *n*. This argument should be a **1** or a **2** referring to either *file1* or *file2*, respectively.

OPTIONS

–a*n*	Produces a line for each unpairable line in file *n*, where *n* is 1 or 2, in addition to the normal output.
–e *s*	Replaces empty output fields by string *s*.
–j*n m*	Joins on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with **1**.
–o *list*	Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.
–t*c*	Uses character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

The following command line joins the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

```
join -j1 4 -j2 3 -o 1.1 2.1 1.6 -t: /etc/passwd /etc/group
```

BUGS

With default field separation, the collating sequence is that of **sort –b**; with **–t**, the sequence is that of a plain sort.

The conventions of **join**, **sort**, **comm**, **uniq** and **awk**(1) are wildly incongruous. Filenames that are numeric may cause conflict when the **–o** *option* is used right before listing filenames.

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

NAME

kbm – set/display keyboard characteristics

SYNOPSIS

kbm [–c *args*] [–l *args*] [–s *args*]

DESCRIPTION

kbm allows you to set the characteristics for the keyboard. Settable characteristics are the compose key(s), and the long and short shift key(s) on the Domain multinational keyboard. The compose key is used to compose characters of the latin-1 character set that do not have corresponding keys on the keyboard. Long and short shift are used to toggle the alternate key labels on the multinational keyboards.

OPTIONS

If no options are specified, **kbm** displays the current keyboard type and characteristics.

–c *args*	Set compose keys to those specified by list *args*.
–l *args*	Set long shift keys to those specified by list *args*.
–s *args*	Set short shift keys to those specified by list *args*.

A key list is a list of function key names separated by commas. The following keys are allowable:

Key Name	**Positions**
`l1-lf`	Left function keys
`r1-r6`	Right function keys
`f0-f9`	Center function keys
`np0-np9, npa-npg, npp`	Numeric pad
`tab,`	TAB
`bs`	BACKSPACE
`ar,al`	ALT keys (multinational keyboard only)

Shifted keys are specified by appending an "s" to the key name, control keys by appending a "c", the up transition by appending an "u"; for example ar, ars, arc, aru .

To disable a function specify a key name of "none".

EXAMPLES

Display current characteristics

```
$ kbm
keyboard:   3
compose:    f5
long_alt:   als,ars
short_alt:  al,ar
```

Set long shift keys to shifted **ar** and shifted **al**; short shift keys to **al** and **ar**.

```
$ kbm –l als,ars –s al,ar
```

Disable the compose function.

```
$ kbm –c none
```

NAME

kill – terminate a process

SYNOPSIS

kill [– signo] PID ...

DESCRIPTION

kill sends signal 15 (terminate) to the specified processes. This kills processes that do not catch or ignore the signal. The process number (PID) of each asynchronous process started with & is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using ps(1).

The details of the kill are described in **kill**(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by – is given as first argument, that signal is sent instead of terminate (see **signal**(2)). In particular **kill –9** is a sure kill.

SEE ALSO

ps(1), sh(1).
kill(2), signal(2) in the *SysV Programmer's Reference*.

NAME

ksh – the Korn shell command programming language

SYNOPSIS

ksh [–aefhikmnoprstuvx] [–o *option* **] . . . [–c** *string* **] [–D** *name=val* **. . .]**
[*arg* **. . .]**

DESCRIPTION

ksh is a command programming language that executes commands read from a terminal or a file. See **Invocation** below for the meaning of arguments to the shell. **rksh** (the restricted version of this shell) is not supported by SysV.

Definitions

A *metacharacter* is one of the following characters:

; & ( ) | < > new-line space tab

A *blank* is a **tab** or a **space**. An *identifier* is a sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for *aliases*, *functions*, and *named parameters*. A *word* is a sequence of *characters* separated by one or more non-quoted *metacharacters*.

Commands

A *simple-command* is a sequence of *blank* separated words which may be preceded by a parameter assignment list. (See **Environment** below). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see exec(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see **signal**(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|**. The standard output of each command but the last is connected by a **pipe**(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **| |**, and optionally terminated by **;**, **&**, or **|&**. Of these five symbols, **;**, **&**, and **|&** have equal precedence, which is lower than that of **&&** and **| |**. The symbols **&&** and **| |** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **|&** causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell. The standard input and output of the spawned command can be written to and read from by the parent shell using the **–p** option of the special commands **read** and **print** described later. Only one such command can be active at any given time. The symbol **&&** (**| |**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) value. An arbitrary number of new-lines may appear in a *list,* instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *identifier* **in** *word* ... **do** *list* **done**

Each time a **for** command is executed, *identifier* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see **Parameter Substitution** below). Execution ends when there are no more words in the list.

select *identifier* **in** *word* ... **do** *list* **done**

A **select** command prints on standard error (file descriptor 2), the set of *word*s, each preceded by a number. If **in** *word* ... is omitted, then the positional parameters are used instead (see **Parameter Substitution** below). The **PS3** prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed **words**, then the value of the parameter *identifier* is set to the *word* corresponding to this number. If this line is empty the selection list is printed again. Otherwise the value of the parameter *identifier* is set to **null**. The contents of the line read from standard input is saved in the parameter **REPLY**. The *list* is executed for each selection until a **break** or *end-of-file* is encountered.

case *word* **in** *pattern* | *pattern* ...) *list* ;; ... **esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see **File Name Generation** below).

if *list* **then** *list* **elif** *list* **then** *list* ... **else** *list* **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**
until *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a separate environment. Note, that if two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below. A parenthesized list used as a command argument denotes *process substitution* as described below.

{ *list*;}

list is simply executed. Note that { is a *keyword* and requires a blank in order to be recognized.

function *identifier* { *list* ;}
identifier () { *list* ;}

Define a function which is referenced by *identifier*. The body of the function is the *list* of commands between { and }. (See **Functions** below).

time *pipeline*

The *pipeline* is executed and the elapsed time as well as the user and system time are printed on standard error.

The following keywords are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { } function select time

Comments

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Aliasing

The first word of each command is replaced by the text of an **alias** if an **alias** for this word has been defined. The first character of an **alias** name can be any non-special printable character, but the rest of the characters must be the same as for a valid *identifier*. The replacement string can contain any valid shell script including the meta-characters listed above. The first word of each command of the replaced text will not be tested for additional aliases. If the last character of the alias value is a *blank* then the word following the alias will also be checked for alias substitution. Aliases can be used to redefine special builtin commands but cannot be used to redefine the keywords listed above. Aliases can be created, listed, and exported with the **alias** command and can be removed with the **unalias** command. Exported aliases remain in effect for sub-shells but must be reinitialized for separate invocations of the shell (See **Invocation** below).

Aliasing is performed when scripts are read, not while they are executed. Therefore, for an alias to take effect the **alias** command has to be executed before the command which references the alias is read.

Aliases are frequently used as a short hand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full pathname of the corresponding command. These aliases are called *tracked* aliases. The value of a *tracked* alias is defined the first time the corresponding command is looked up and becomes undefined each time the **PATH** variable is reset. These aliases remain *tracked* so that the next subsequent reference will redefine the value. Several tracked aliases are compiled into the shell. The **−h** option of the **set** command makes each command name which is a valid alias name into a tracked alias.

The following *exported aliases* are compiled into the shell but can be unset or redefined:

```
false='let 0'
functions='typeset -f'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
hash='alias -t'
```

Tilde Substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~. If it does, then the word up to a / is checked to see if it matches a user name in the /etc/**passwd** file. If a match is found, the ~ and the matched login name is replaced by the login directory of the matched user. This is called a *tilde* substitution. If no match is found, the original text is left unchanged. A ~ by itself, or in front of a /, is replaced by the value of the **HOME** parameter. A ~ followed by a + or – is replaced by the value of the parameter **PWD** and **OLDPWD** respectively.

In addition, the value of each *keyword parameter* is checked to see if it begins with a ~ or if a ~ appears after a :. In either of these cases a *tilde* substitution is attempted.

Command Substitution

The standard output from a command enclosed in parenthesis preceded by a dollar sign ($()) or a pair of grave accents (\`\`) may be used as part or all of a word; trailing newlines are removed. In the second (archaic) form, the string between the quotes is processed for special quoting characters before the command is executed. (See **Quoting** below). The command substitution **$(cat file)** can be replaced by the equivalent but faster **$(<file)**. Command substitution of most special commands that do not perform input/output redirection are carried out without creating a separate process.

Parameter Substitution

A *parameter* is an *identifier*, one or more digits, or any of the characters *****, **@**, **#**, **?**, **–**, **$**, and **!**. A *named parameter* (a parameter denoted by an identifier) has a *value* and zero or more *attributes*. *Named parameters* can be assigned *values* and *attributes* by using the **typeset** special command. The attributes supported by the shell are described later with the **typeset** special command. Exported parameters pass values and attributes to sub-shells but only values to the environment.

The shell supports a limited one-dimensional array facility. An element of an array parameter is referenced by a *subscript*. A *subscript* is denoted by a [, followed by an *arithmetic expression* (see Arithmetic evaluation below) followed by a]. The value of all subscripts must be in the range of 0 through 511. Arrays need not be declared. Any reference to a named parameter with a valid subscript is legal and an array will be created if necessary. Referencing an array without a subscript is equivalent to referencing the first element.

The *value* of a *named parameter* may also be assigned by writing:

> *name*=*value*
>
> *name*=*value*
>
> ...

If the integer attribute, **–i**, is set for *name* the *value* is subject to arithmetic evaluation as described below.
Positional parameters, parameters denoted by a number, may be assigned values with the set special command. Parameter **$0** is set from argument zero when the shell is invoked.
The character $ is used to introduce substitutable *parameters*.

${*parameter*}
: The value, if any, of the parameter is substituted. The braces are required when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name or when a named parameter is subscripted. If *parameter* is one or more digits then it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If *parameter* is * or @, then all the positional parameters, starting with **$1**, are substituted (separated by a field separator character). If an array *identifier* with subscript * or @ is used, then the value for each of the elements is substituted (separated by a field separator character).

${#*parameter*}
: If *parameter* is * or @, the number of positional parameters is substituted. Otherwise, the length of the value of the *parameter* is substituted.

${#*identifier*[*]}
: The number of elements in the array *identifier* is substituted.

${*parameter*:–*word*}
: If *parameter* is set and is non-null then substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
: If *parameter* is not set or is null then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
: If *parameter* is set and is non-null then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted then a standard message is printed.

${*parameter*:+*word*}
: If *parameter* is set and is non-null then substitute *word*; otherwise substitute nothing.

${*parameter*#*pattern*}
${*parameter*##*pattern*}
: If the shell *pattern* matches the beginning of the value of *parameter*, then the value of this substitution is the value of the *parameter* with the matched portion deleted; otherwise the value of this *parameter* is substituted. In the first form the smallest matching pattern is deleted and in the latter form the largest matching pattern is deleted.

${*parameter*%*pattern*}
${*parameter*%%*pattern*}
: If the shell *pattern* matches the end of the value of *parameter*, then the value of *parameter* with the matched part deleted; otherwise substitute the value of *parameter*. In the first form the smallest matching pattern is deleted and in the latter form the largest matching pattern is deleted.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:–$(pwd)}
```

If the colon (:) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
–	Flags supplied to the shell on invocation or by the set command.
?	The decimal value returned by the last executed command.
$	The process number of this shell.
_	The last argument of the previous command. This parameter is not set for commands which are asynchronous. This parameter is also used to hold the name of the matching **MAIL** file when checking for mail. Finally, the value of this parameter is set to the full pathname of each program the shell invokes and is passed in the *environment*.
!	The process number of the last background command invoked.
PPID	The process number of the parent of the shell.
PWD	The present working directory set by the **cd** command.
OLDPWD	The previous working directory set by the **cd** command.
RANDOM	Each time this parameter is referenced, a random integer is generated. The sequence of random numbers can be initialized by assigning a numeric value to **RANDOM**.

REPLY This parameter is set by the **select** statement and by the **read** special command when no arguments are supplied.

SECONDS
Each time this parameter is referenced, the number of seconds since shell invocation is returned. If this parameter is assigned a value, then the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

The following parameters are used by the shell:

CDPATH
The search path for the *cd* command.

COLUMNS
If this variable is set, the value is used to define the width of the edit window for the shell edit modes and for printing **select** lists.

EDITOR
If the value of this variable ends in **emacs**, **gmacs**, or **vi** and the **VISUAL** variable is not set, then the corresponding option (see Special Command **set** below) will be turned on. This value should be unset for shells running in Apollo transcript pads.

ENV If this parameter is set, then parameter substitution is performed on the value to generate the pathname of the script that will be executed when the shell is invoked. (See **Invocation** below.) This file is typically used for *alias* and *function* definitions.

FCEDIT
The default editor name for the fc command. In Apollo transcript pads, this variable should be set to 'pad'. On dialup lines or in VT100 windows, values like 'vi' or 'emacs' are useful.

IFS Internal field separators, normally **space**, **tab**, and **new-line** that is used to separate command words which result from command or parameter substitution and for separating words with the special command **read**. The first character of the **IFS** parameter is used to separate arguments for the **"$*"** substitution (See **Quoting** below).

HISTFILE
If this parameter is set when the shell is invoked, then the value is the pathname of the file that will be used to store the command history. (See **Command Re-entry** below.)

HISTSIZE
If this parameter is set when the shell is invoked, then the number of previously entered commands that are accessible by this shell will be greater than or equal to this number. The default is 128.

HOME The default argument (home directory) for the **cd** command.

LINES If this variable is set, the value is used to determine the column length for printing **select** lists. Select lists will print vertically until about two-thirds of **LINES** lines are filled.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, then the shell informs the user of arrival of mail in the specified file.

MAILCHECK
This variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds. When the time has elapsed the shell will check before issuing the next prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set then the shell informs the user of any modifications to the specified files that have occurred within the last **MAILCHECK** seconds. Each file name can be followed by a ? and a message that will be printed. The message will undergo parameter and command substitution with the parameter, $_ defined as the name of the file that has changed. The default message is *you have mail in $_*.

PATH The search path for commands (see **Execution** below). The user may not change **PATH** if executing under **rksh** (except in *.profile*).

PS1 The value of this parameter is expanded for parameter substitution to define the primary prompt string which by default is "$ ". The character ! in the primary prompt string is replaced by the *command* number (see **Command Re-entry** below).

PS2 Secondary prompt string, by default "> ".

PS3 Selection prompt string used within a select loop, by default "#? ".

SHELL The pathname of the *shell* is kept in the environment. This value should be unset for shells running in Apollo transcript pads.

TMOUT
If set to a value greater than zero, the shell will terminate if a command is not entered within the prescribed number of seconds afte issuing the **PS1** prompt. (Note that the shell can be compiled with maximum bound for this value which cannot be exceeded.)

VISUAL
If the value of this variable ends in **emacs**, **gmacs**, or **vi** then th corresponding option (see Special Command set below) will b turned on.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK**, **TMOUT** and **IFS** while **HOME**, **SHELL ENV** and **MAIL** are not set at all by the shell (although **HOME** *i* set by **login**(1)). On some systems **MAIL** and **SHELL** are also set by **login**(1)).

Blank Interpretation

After parameter and command substitution, the results of substitutions are scanned for the field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation

Following substitution, each command *word* is scanned for the characters *, ?, and unless the **-f** option has been set. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. When a *pattern* is used for file name generation, the character **.** at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly. In other instances of pattern matching the / and **.** are not treated specially.

*	Matches any string, including the null string.
?	Matches any single character.
...	Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "! " then any character not enclosed is matched. A – can be included in the character set by putting it as the first or last character.

Quoting

Each of the *metacharacters* listed above (See **Definitions** above) has a special meaning to the shell and causes termination of a word unless quoted. A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **new-line** is ignored. All characters enclosed between a pair of single quote marks (′′), are quoted. A single quote cannot appear within single quotes. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, ˋ, ", and $. The meaning of $* and $@ is identical when not quoted or when used as a parameter assignment value or as a file name. However, when used as a command argument, "$*" is equivalent to "$1*d*$2*d*...", where *d* is the first character of the **IFS** parameter, whereas "$@" is equivalent to "$1" "$2" .... Inside grave quote marks (ˋˋ) \ quotes the characters \, ˋ, and $. If the grave quotes occur within double quotes then \ also quotes the character ".

The special meaning of keywords or aliases can be removed by quoting any character of the keyword. The recognition of function names or special command names listed below cannot be altered by quoting them.

Arithmetic Evaluation

An ability to perform integer arithmetic is provided with the special command **let**. Evaluations are performed using *long* arithmetic. Constants are of the form ***base#n*** where *base* is a decimal number between two and thirty-six representing the arithmetic base and *n* is a number in that base. If *base* is omitted then base 10 is used.

An internal integer representation of a *named parameter* can be specified with the **–i** option of the **typeset** special command. When this attribute is selected the first assignment to the parameter determines the arithmetic base to be used when parameter substitution occurs.

Since many of the arithmetic operators require quoting, an alternative form of the **let** command is provided. For any command which begins with a ((, all the characters until a matching)) are treated as a quoted expression. More precisely, ((...)) is equivalent to let "...".

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of **PS2**) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command. Command and parameter substitution occurs before *word* or *digit* is used except as noted below. File name generation occurs only if the pattern matches a single file and blank interpretation is not performed.

<*word*	Use file *word* as standard input (file descriptor 0).
>*word*	Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.
>>*word*	Use file *word* as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
<<–*word*	The shell input is read up to a line that is the same as *word*, or to an end-of-file. No parameter substitution, command substitution or file name generation is performed on *word*. The resulting document, called a *here-document*, becomes the standard input. If any character of *word* is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, \new-line is ignored, and \\ must be used to quote the characters \\, $, \`, and the first character of *word*. If – is appended to <<, then all leading tabs are stripped from *word* and from the document.

<&*digit*	The standard input is duplicated from file descriptor *digit* (see **dup**(2)). Similarly for the standard output using >& *digit*.
<&−	The standard input is closed. Similarly for the standard output using >&−.

If one of the above is preceded by a digit, then the file descriptor number referred to is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (*file descriptor*, *file*) association at the time of evaluation. For example:

... 1>*fname* 2>&1

first associates file descriptor 1 with file *fname*. It then associates file descriptor 2 with the file associated with file descriptor 1 (i.e. *fname*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be associated with file *fname*.

If a command is followed by & and job control is not active, then the default standard input for the command is the empty file /dev/**null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The *environment* (see **environ**(7)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The names must be *identifiers* and the values are character strings. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value and marking it *export*. Executed commands inherit the environment. If the user modifies the values of these parameters or creates new ones, using the **export** or **typeset** −x commands they become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions which must be noted in **export** or **typeset** −x commands.

The environment for any *simple-command* or function may be augmented by prefixing it with one or more parameter assignments. A parameter assignment argument is a word of the form *identifier=value*. Thus:

```
TERM=450 cmd args                          and
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned).

If the −k flag is set, *all* parameter assignment arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c:**

```
echo a=b c
set −k
echo a=b c
```

Functions

The **function** keyword, described in the *Commands* section above, is used to define shell functions. Shell functions are read in and stored internally. Alias names are resolved when the function is read. Functions are executed like commands with the arguments passed as positional parameters. (See **Execution** below).

Functions execute in the same process as the caller and share all files, traps (other than **EXIT** and **ERR**) and present working directory with the caller. A trap set on **EXIT** inside a function is executed after the function completes. Ordinarily, variables are shared between the calling program and the function. However, the **typeset** special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command **return** is used to return from function calls. Errors within functions return control to the caller.

Function identifiers can be listed with the **−f** option of the **typeset** special command. The text of functions will also be listed. Function can be undefined with the **−f** option of the **unset** special command.

Ordinarily, functions are unset when the shell executes a shell script. The **−xf** option of the **typeset** command allows a function to be exported to scripts that are executed without a separate invocation of the shell. Functions that need to be defined across separate invocations of the shell should be placed in the **ENV** file.

Jobs

If the **monitor** option of the **set** command is turned on, an interactive shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers. When a job is started asynchronously with **&**, the shell prints a line which looks like:

```
[1] 1234
```

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

This paragraph and the next require features that are not in all versions of UNIX and may not apply. If you are running a job and wish to do something else you can press **CTRL/Z** (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been 'Stopped', and print another prompt. You can then manipulate the state of this job, putting it in the background with the **bg** command, or run some other commands and then eventually bring the job back into the foreground with the foreground command **fg**. A **^Z** takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command "stty tostop". If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. The character % introduces a job name. If you wish to refer to job number 1, you can name it as %**1** . Jobs can also be named by prefixes of the string typed in to **kill** or restart them. Thus, on systems that support job control, '**fg** %**ed**' would normally restart a suspended **ed**(1) job, if there were a suspended job whose name began with the string 'ed'.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a + and the previous job with a –. The abbreviation %+ refers to the current job and %– refers to the previous job. %% is also a synonym for the current job.

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work.

When you try to leave the shell while jobs are running or stopped, you will be warned that 'You have stopped(running) jobs.' You may use the **jobs** command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated.

Signals

The INT and QUIT signals for an invoked command are ignored if the command is followed by & and job **monitor** option is not active. Otherwise, signals have the values inherited by the shell from its parent (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed within the current shell process. Next, the command name is checked to see if it matches one of the user defined functions. If it does, the positional parameters are saved and then reset to the arguments of the *function* call. When the *function* completes or issues a **return**, the positional parameter list is restored and any trap set on **EXIT** within the function is executed. The value of a *function* is the value of the last command executed. A function is also executed in the current shell process. If a command name is not a *special command* or a user defined *function*, a process is created and an attempt is made to execute the command via exec(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **/bin:/usr/bin:** (specifying **/bin**, **/usr/bin**, and the current directory in that order). The current directory can be specified by two or more adjacent colons, or by a colon at the beginning or end of the path list. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. All non-exported aliases, functions, and named parameters are removed in this case. If the shell command file doesn't have read permission, or if the *setuid* and/or *setgid* bits are set on the file, then the shell executes an agent whose job it is to set up the permissions and execute the shell with the shell command file passed down as an open file. A parenthesized command is also executed in a sub-shell without removing non-exported quantities.

Command Re-entry

The text of the last **HISTSIZE** (default 128) commands entered from a terminal device is saved in a *history* file. The file **$HOME/.sh_history** is used if the **HISTFILE** variable is not set or is not writable. A shell can access the commands of all *interactive* shells which use the same named **HISTFILE**. The special command fc is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified. If you do not specify an editor program as an argument to **fc** then the value of the parameter **FCEDIT** is used. If **FCEDIT** is not defined then **/bin/ed** is used. The edited command(s) is printed and re-executed upon leaving the editor. The editor name – is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form *old=new* can be used to modify the command before execution. For example, if **r** is aliased to **′fc –e –′** then typing ‘**r bad=good c**’ will re-execute the most recent command which starts with the letter **c**, replacing the first occurrence of the string **bad** with the string **good**.

In-line Editing Options

Normally, each command line entered from a terminal device is simply typed followed by a new-line (‘RETURN’ or ‘LINE FEED’). If either the **emacs**, **gmacs**, or **vi** option is active, the user can edit the command line. To be in either of these edit modes set the corresponding option. An editing option is automatically selected each time the **VISUAL** or **EDITOR** variable is assigned a value ending in either of these option names.

The editing features require that the user's terminal accept ‘RETURN’ as carriage return without line feed and that a space (‘ ’) must overwrite the current character on the screen. ADM terminal users should set the "space - advance" switch to ‘space’. Hewlett-Packard series 2621 terminal users should set the straps to ‘bcGHxZ etX’.

The editing modes implement a concept where the user is looking through a window at the current line. The window width is the value of **COLUMNS** if it is defined, otherwise 80. If the line is longer than the window width minus two, a mark is displayed at the end of the window to notify the user. As the cursor moves and reaches the window boundaries the window will be centered about the cursor. The mark is a > (<, *) if the line extends on the right (left, both) side(s) of the window.

The in-line editing options are not useful in Apollo transcript pads. The command input pane associated with transcript pads allows full command line editing. Setting **VISUAL** or **EDITOR** in Apollo transcript pads causes the pad to flip in and out of raw mode.

In-Line editing is very useful on dialup UP terminals or in a VT100 window where no other editing is available.

Emacs Editing Mode

This mode is entered by enabling either the **emacs** or **gmacs** option. The only difference between these two modes is the way they handle **^T**. To edit, the user moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. All the editing commands are control characters or escape sequences. The notation for control characters is caret (^) followed by the character. For example, **^F** is the notation for control **F**. This is entered by pressing 'f' while holding down the 'CTRL' (control) key. The 'SHIFT' key is *not* pressed. (The notation **^?** indicates the DEL (delete) key.)

The notation for escape sequences is **M-** followed by a character. For example, **M-f** (pronounced Meta f) is entered by depressing ESC (ascii 033) followed by 'f'. (**M-F** would be the notation for ESC followed by 'SHIFT' (capital) 'F'.)

All edit commands operate from any place on the line (not just at the beginning). Neither the "RETURN" nor the "LINE FEED" key is entered after edit commands except when noted.

^F	Move cursor forward (right) one character.
M-f	Move cursor forward one word. (The editor's idea of a word is a string of characters consisting of only letters, digits and underscores.)
^B	Move cursor backward (left) one character.
M-b	Move cursor backward one word.
^A	Move cursor to start of line.
^E	Move cursor to end of line.
^]*char*	Move cursor to character *char* on current line.
^X^X	Interchange the cursor and mark.
erase	(User defined erase character as defined by the stty command, usually **^H** or #.) Delete previous character.
^D	Delete current character.
M-d	Delete current word.
M-^H	(Meta-backspace) Delete previous word.
M-h	Delete previous word.
M-^?	(Meta-DEL) Delete previous word (if your interrupt character is **^?** (DEL, the default) then this command will not work).

^T	Transpose current character with next character in *emacs* mode. Transpose two previous characters in *gmacs* mode.
^C	Capitalize current character.
M-c	Capitalize current word.
M-l	Change the current word to lower case.
^K	Kill from the cursor to the end of the line. If given a parameter of zero then kill from the start of line to the cursor.
^W	Kill from the cursor to the mark.
M-p	Push the region from the cursor to the mark on the stack.
kill	(User defined kill character as defined by the stty command, usually ^G or @.) Kill the entire current line. If two *kill* characters are entered in succession, all kill characters from then on cause a line feed (useful when using paper terminals).
^Y	Restore last item removed from line. (Yank item back to the line.)
^L	Line feed and print current line.
^@	(Null character) Set mark.
M-	(Meta space) Set mark.
^J	(New line) Execute the current line.
^M	(Return) Execute the current line.
eof	End-of-file character, normally **^D**, will terminate the shell if the current line is null.
^P	Fetch previous command. Each time **^P** is entered the previous command back in time is accessed.
M-<	Fetch the least recent (oldest) history line.
M->	Fetch the most recent (youngest) history line.
^N	Fetch next command. Each time **^N** is entered the next command forward in time is accessed.
^R*string*	Reverse search history for a previous command line containing *string*. If a parameter of zero is given, the search is forward. *String* is terminated by a "RETURN" or "NEW LINE". If *string* is omitted, then the next command line containing the most recent *string* is accessed. In this case a parameter of zero reverses the direction of the search.
^O	Operate – Execute the current line and fetch the next line relative to current line from the history file.
M-*digits*	(Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a parameter are **.**, **^F**, **^B**, *erase*, **^D**, **^K**, **^R**, **^P**, **^N**, **M-.**, **M-_**, **M-b**, **M-c**, **M-d**, **M-f**, **M-h** and **M-^H**.
M-*letter*	Soft-key – Your alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value will be inserted on the input queue. The *letter* must not be one of the above meta-functions.
M-.	The last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.

M-_	Same as M-..
M-*	Attempt file name generation on the current word. An asterisk is appended if the word doesn't contain any special pattern characters.
M-ESC	Same as M-*.
M-=	List files matching current word pattern if an asterisk were appended.
^U	Multiply parameter of next command by 4.
\	Escape next character. Editing characters, the user's erase, kill and interrupt (normally ^?) characters may be entered in a command line or in a search string if preceded by a \. The \ removes the next character's editing features (if any).
^V	Display version of the shell.

Vi Editing Mode

There are two typing modes. Initially, when you enter a command you are in the *input* mode. To edit, the user enters *control* mode by typing ESC (033) and moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. Most control commands accept an optional repeat *count* prior to the command. When in vi mode on most systems, canonical processing is initially enabled and the command will be echoed again if the speed is 1200 baud or greater and it contains any control characters or less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command and the user can than modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode. If the option viraw is also set, the terminal will always have canonical processing disabled. This mode is implicit for systems that do not support two alternate end of line delimiters, and may be helpful for certain terminals.

Input Edit Commands

By default the editor is in input mode.

erase	(User defined erase character as defined by the stty command, usually ^H or #.) Delete previous character.
^W	Delete the previous blank separated word.
^D	Terminate the shell.
^V	Escape next character. Editing characters, the user's erase or kill characters may be entered in a command line or in a search string if preceded by a ^V. The ^V removes the next character's editing features (if any).
\	Escape the next *erase* or *kill* character.

Motion Edit Commands

These commands will move the cursor.

[*count*]l	Cursor forward (right) one character.
[*count*]w	Cursor forward one alpha-numeric word.
[*count*]W	Cursor to the beginning of the next word that follows a blank.
[*count*]e	Cursor to end of word.

[*count*]E	Cursor to end of the current blank delimited word.
[*count*]**h**	Cursor backward (left) one character.
[*count*]**b**	Cursor backward one word.
[*count*]**B**	Cursor to preceding blank separated word.
[*count*]**f***c*	Find the next character *c* in the current line.
[*count*]F*c*	Find the previous character *c* in the current line.
[*count*]**t***c*	Equivalent to **f** followed by **h**.
[*count*]T*c*	Equivalent to F followed by **l**.
;	Repeats the last single character find command, **f**, **F**, **t**, or **T**.
,	Reverses the last single character find command.
0	Cursor to start of line.
^	Cursor to first non-blank character in line.
$	Cursor to end of line.

Search Edit Commands

These commands access your command history.

[*count*]**k**	Fetch previous command. Each time **k** is entered the previous command back in time is accessed.
[*count*]–	Equivalent to **k**.
[*count*]**j**	Fetch next command. Each time **j** is entered the next command forward in time is accessed.
[*count*]+	Equivalent to **j**.
[*count*]G	The command number *count* is fetched. The default is the least recent history command.
/*string*	Search backward through history for a previous command containing *string*. *String* is terminated by a "RETURN" or "NEW LINE". If *string* is null the previous string will be used.
?*string*	Same as / except that search will be in the forward direction.
n	Search for next match of the last pattern to / or ? commands.
N	Search for next match of the last pattern to / or ?, but in reverse direction. Search history for the *string* entered by the previous / command.

Text Modification Edit Commands

These commands will modify the line.

a	Enter input mode and enter text after the current character.
A	Append text to the end of the line. Equivalent to **$a**.
[*count*]c*motion* c[*count*]*motion*	Delete current character through the character that *motion* would move the cursor to and enter input mode. If *motion* is c, the entire line will be deleted and input mode entered.
C	Delete the current character through the end of line and enter input mode. Equivalent to c**$**.
S	Equivalent to cc.

D Delete the current character through the end of line. Equivalent to **d$**.

[*count*]d*motion*

d[*count*]*motion*

Delete current character through the character that *motion* would move to. If *motion* is **d** , the entire line will be deleted.

i Enter input mode and insert text before the current character.

I Insert text before the beginning of the line. Equivalent to the two character sequence ^**i.**

[*count*]**P** Place the previous text modification before the cursor.

[*count*]**p** Place the previous text modification after the cursor.

R Enter input mode and replace characters on the screen with characters you type overlay fashion.

r*c* Replace the current character with *c*.

[*count*]x Delete current character.

[*count*]X Delete preceding character.

[*count*]. Repeat the previous text modification command.

~ Invert the case of the current character and advance the cursor.

[*count*]_ Causes the *count* word of the previous command to be appended and input mode entered. The last word is used if *count* is omitted.

* Causes an * to be appended to the current word and file name generation attempted. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.

Other Edit Commands

Miscellaneous commands.

[*count*]y*motion*

y[*count*]*motion*

Yank current character through character that *motion* would move the cursor to and puts them into the delete buffer. The text and cursor are unchanged.

Y Yanks from current position to end of line. Equivalent to y$.

u Undo the last text modifying command.

U Undo all the text modifying commands performed on the line.

[*count*]v Returns the command **fc -e ${VISUAL:-${EDITOR:-vi}}** *count* in the input buffer. If *count* is omitted, then the current line is used.

^L Line feed and print current line. Has effect only in control mode.

^J (New line) Execute the current line, regardless of mode.

^M (Return) Execute the current line, regardless of mode.

Sends the line after inserting a # in front of the line and after each new-line. Useful for causing the current line to be inserted in the history without being executed.

=	List the file names that match the current word if an asterisk were appended it.
@letter	Your alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value will be inserted on the input queue for processing.

Special Commands

The following simple-commands are executed in the shell process. Input/Output redirection is permitted. Unless otherwise indicated, the output is written on file descriptor 1. Commands that are preceded by one or two † are treated specially in the following ways:

1. Parameter assignment lists preceding the command remain in effect when the command completes.
2. They are executed in a separate process when used within command substitution.
3. Errors in commands preceded by †† cause the script that contains them to abort.

† : *arg* ...

The command only expands parameters. A zero exit code is returned.

†† . *file arg* ...

Read and execute commands from *file* and return. The commands are executed in the current shell environment. The search path specified by **PATH** is used to find the directory containing *file*. If any arguments *arg* are given, they become the positional parameters. Otherwise the positional parameters are unchanged.

alias **–tx** *name* =*value* ...

Alias with no arguments prints the list of aliases in the form *name*=*value* on standard output. An *alias* is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution. The –t flag is used to set and list tracked aliases. The value of a tracked alias is the full pathname corresponding to the given *name*. The value becomes undefined when the value of **PATH** is reset but the aliases remained tracked. Without the –t flag, for each *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The –x flag is used to set or print exported aliases. An exported alias is defined across sub-shell environments. Alias returns true unless a *name* is given for which no alias has been defined.

bg *%job*

This command is only built-in on systems that support job control. Puts the specified *job* into the background. The current job is put in the background if *job* is not specified.

break *n*
: Exit from the enclosing **for while until** or **select** loop, if any. If *n* is specified then break *n* levels.

continue *n*
: Resume the next iteration of the enclosing **for while until** or **select** loop. If *n* is specified then resume at the *n*-th enclosing loop.

† **cd** *arg*
† **cd** *old new*
: This command can be in either of two forms. In the first form it changes the current directory to *arg*. If *arg* is – the directory is changed to the previous directory. The shell parameter **HOME** is the default *arg*. The parameter **PWD** is set to the current directory. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / then the search path is not used. Otherwise, each directory in the path is searched for *arg*. The second form of **cd** substitutes the string *new* for the string *old* in the current directory name, **PWD** and tries to change to this new directory. The **cd** command may not be executed by **rksh**.

echo *arg* ...
: See **echo**(1) for usage and description.

†† **eval** *arg* ...
: The arguments are read as input to the shell and the resulting command(s) executed.

†† **exec** *arg* ...
: If *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and affect the current process. If no arguments are given the effect of this command is to modify file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 that are opened with this mechanism are closed when invoking another program.

exit *n*
: Causes the shell to exit with the exit status specified by *n*. If *n* is omitted then the exit status is that of the last command executed. An end-of-file will also cause the shell to exit except for a shell which has the **ignoreeof** option (See **set** below) turned on.

†† **export** *name* ...
: The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands.

†† **fc** **−e** *ename* **−nlr** *first* *last*
†† **fc** **−e** **−** *old=new* *command*

In the first form, a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands that were typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the flag **−l**, is selected, the commands are listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, then the value of the parameter **FCEDIT** (default /bin/ed) is used as the editor. When editing is complete, the edited command(s) is executed. If *last* is not specified then it will be set to *first*. If *first* is not specified the default is the previous command for editing and −16 for listing. The flag **−r** reverses the order of the commands and the flag **−n** suppresses command numbers when listing. In the second form the *command* is re-executed after the substitution *old=new* is performed.

fg *%job*

This command is only built-in on systems that support job control. If *job* is specified it brings it to the foreground. Otherwise, the current job is brought into the foreground.

inlib *pathname*

Install a user-supplied library specified by *pathname* in the current (shell) process. The library is used to resolve external references of programs (and libraries) loaded after its installation. Note that the library is not loaded into the address space unless it is needed to resolve an external reference. The list of inlibed libraries is passed to all children of the current shell. Use **llib**(1) to examine this list.

jobs **−l** Lists the active jobs; given the **−l** options lists process id's in addition to the normal information.

kill *−sig process* ...

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in **/usr/include/signal.h**, stripped of the prefix "SIG"). The signal numbers and names are listed by '**kill −l**'. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal if it is stopped. The argument *process* can be either a process id or a job.

let *arg* ...

Each *arg* is an *arithmetic expression* to be evaluated. All calculations are done as long integers and no check for overflow is performed. Expressions consist of constants, named parameters, and operators.

The following set of operators, listed in order of decreasing precedence, have been implemented:

−	unary minus
!	logical negation
* / %	multiplication, division, remainder
+ −	addition, subtraction
<= >= < >	comparison
== !=	equality inequality
=	arithmetic replacement

Sub-expressions in parentheses ( ) are evaluated first and can be used to override the above precedence rules. The evaluation within a precedence group is from right to left for the = operator and from left to right for the others.

A parameter name must be a valid *identifier*. When a parameter is encountered, the value associated with the parameter name is substituted and expression evaluation resumes. Up to nine levels of recursion are permitted.

The return code is 0 if the value of the last expression is non-zero, and 1 otherwise.

†† **newgrp** *arg* ...
Equivalent to exec **newgrp** *arg* ....

print −Rnprsu*n* *arg* ...
The shell output mechanism. With no flags or with flag −, the arguments are printed on standard output as described by **echo**(1). In raw mode, **−R** or **−r**, the escape conventions of **echo** are ignored. The **−R** option will print all subsequent arguments and options other than **−n**. The **−p** option causes the arguments to be written onto the pipe of the process spawned with | & instead of standard output. The −s option causes the arguments to be written onto the history file instead of standard output. The **−u** flag can be used to specify a one digit file descriptor unit number **n** on which the output will be placed. The default is 1. If the flag **−n** is used, no **new-line** is added to the output.

pwd Equivalent to **print −r − $PWD**

read −prsu *n* *name?prompt* *name* ...
The shell input mechanism. One line is read and is broken up into words using the characters in **IFS** as separators. In raw mode, **−r**, a \ at the end of a line does not signify line continuation. The first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The **−p** option causes the input line to be taken from the input pipe of a process spawned by the shell using | &. If the −s flag is present, the input will be saved as a command in the history file. The flag **−u** can be used to

specify a one digit file descriptor unit to read from. The file descriptor can be opened with the exec special command. The default value of *n* is 0. If *name* is omitted then **REPLY** is used as the default *name*. The return code is 0 unless an end-of-file is encountered. An end-of-file with the **−p** option causes cleanup for this process so that another can be spawned. If the first argument contains a ?, the remainder of this word is used as a *prompt* when the shell is interactive. If the given file descriptor is open for writing and is a terminal device then the prompt is placed on this unit. Otherwise the prompt is issued on file descriptor 2. The return code is 0 unless an end-of-file is encountered.

†† **readonly** *name* ...

The given *names* are marked readonly and these names cannot be changed by subsequent assignment.

†† **return** *n*

Causes a shell *function* to return to the invoking script with the return status specified by *n*. If *n* is omitted then the return status is that of the last command executed. If **return** is invoked while not in a *function* or a . script, then it is the same as an **exit**.

rootnode [*arg*]

Change the current node entry directory to *arg*.

set −aefhkmnostuvx **−o** *option* ... *arg* ...

The flags for this command have meaning as follows:

−a	All subsequent parameters that are defined are automatically exported.
−e	If the shell is non-interactive and if a command fails, execute the **ERR** trap, if set, and exit immediately. This mode is disabled while reading profiles.
−f	Disables file name generation.
−h	Each command whose name is an *identifier* becomes a tracked alias when first encountered.
−k	All parameter assignment arguments are placed in the environment for a command, not just those that precede the command name.
−m	Background jobs will run in a separate process group and a line will print upon completion. The exit status of background jobs is reported in a completion message. On systems with job control, this flag is turned on automatically for interactive shells.
−n	Read commands but do not execute them. Ignored for interactive shells.
−o	The following argument can be one of the following option names:

allexport	Same as **−a**.
errexit	Same as **−e**.

bgnice	All background jobs are run at a lower priority.
emacs	Puts you in an *emacs* style in-line editor for command entry.
gmacs	Puts you in a *gmacs* style in-line editor for command entry.
ignoreeof	The shell will not exit on end-of-file. The command **exit** must be used.
keyword	Same as **–k**.
markdirs	All directory names resulting from file name generation have a trailing / appended.
monitor	Same as **–m**.
noexec	Same as **–n**.
noglob	Same as **–f**.
nounset	Same as **–u**.
protected	Same as **–p**.
verbose	Same as **–v**.
trackall	Same as **–h**.
vi	Puts you in insert mode of a *vi* style in-line editor until you hit escape character **033**. This puts you in move mode. A return sends the line.
viraw	Each character is processed as it is typed in *vi* mode.
xtrace	Same as **–x**.

If no option name is supplied then the current option settings are printed.

–p	Resets the **PATH** variable to the default value, disables processing of the **$HOME/.profile** file and uses the file /etc/**suid_profile** instead of the **ENV** file. This mode is automatically enabled whenever the effective uid (gid) is not equal to the real uid (gid).
–s	Sort the positional parameters.
–t	Exit after reading and executing one command.
–u	Treat unset parameters as an error when substituting.
–v	Print shell input lines as they are read.
–x	Print commands and their arguments as they are executed.
–	Turns off –x and –v flags and stops examining arguments for flags.
– –	Do not change any of the flags; useful in setting **$1** to a value beginning with –. If no arguments follow this flag then the positional parameters are unset.

Using + rather than – causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in $–. The remaining arguments are positional parameters and are assigned, in order, to **$1** **$2** .... If no arguments are given, then the values of all names are printed on the standard output.

† **shift** *n*

The positional parameters from $*n*+1 ... are renamed **$1** ... , default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a non-negative number less than or equal to $#.

test *expr*

Evaluate conditional expression *expr*. See **test**(1) for usage and description. The arithmetic comparison operators are not restricted to integers. They allow any arithmetic expression. Four additional primitive expressions are allowed:

−L *file* True if *file* is a symbolic link.

file1 **−nt** *file2*

True if *file1* is newer than *file2*.

file1 −ot *file2*

True if *file1* is older than *file2*.

file1 −ef *file2*

True if *file1* has the same device and i-node number as *file2*.

times Print the accumulated user and system times for the shell and for processes run from the shell.

trap *arg sig* ...

arg is a command to be read and executed when the shell receives signal(s) *sig*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be given as a number or as the name of the signal. Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. If *arg* is omitted or is −, then all trap(s) *sig* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by the commands it invokes. If *sig* is **ERR** then *arg* will be executed whenever a command has a non-zero exit code. This trap is not inherited by functions. If *sig* is **0** or **EXIT** and the **trap** statement is executed inside the body of a function, then the command *arg* is executed after the function completes. If *sig* is **0** or **EXIT** for a **trap** set outside any function then the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

†† **typeset −HLRZfilprtux***n* *name =value* ...

When invoked inside a function, a new instance of the parameter *name* is created. The parameter value and type are restored when the function completes. The following list of attributes may be specified:

−H This flag provides UNIX to host-name file mapping on non-UNIX machines.

–L	Left justify and remove leading blanks from *value*. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. When the parameter is assigned to, it is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the –Z flag is also set. The –**R** flag is turned off.
–R	Right justify and fill with leading blanks. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. The field is left filled with blanks or truncated from the end if the parameter is reassigned. The **L** flag is turned off.
–Z	Right justify and fill with leading zeros if the first non-blank character is a digit and the –L flag has not been set. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment.
–f	The names refer to function names rather than parameter names. No assignments can be made and the only other valid flags are –t, which turns on execution tracing for this function and –x, to allow the function to remain in effect across shell procedures executed in the same process environment.
–i	Parameter is an integer. This makes arithmetic faster. If *n* is non-zero it defines the output arithmetic base, otherwise the first assignment determines the output base.
–l	All upper-case characters converted to lower-case. The upper-case flag, **–u** is turned off.
–p	The output of this command, if any, is written onto the two-way pipe
–r	The given *names* are marked readonly and these names cannot be changed by subsequent assignment.
–t	Tags the named parameters. Tags are user definable and have no special meaning to the shell.
–u	All lower-case characters are converted to upper-case characters. The lower-case flag, –l is turned off.
–x	The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands.

Using + rather than – causes these flags to be turned off. If no *name* arguments are given but flags are specified, a list of *names* (and optionally the *values*) of the *parameters* which have these flags set is printed. (Using + rather than – keeps the values to be printed.) If no *name*s and flags are given, the *names* and *attributes* of all *parameters* are printed.

ulimit –acdfmpst *n*

–a	Lists all of the current resource limits (BSD only).
–d	imposes a size limit of *n* kbytes on the size of the data area (BSD only).

−f	imposes a size limit of *n* 512 byte blocks on files written by child processes (files of any size may be read).
−m	imposes a soft limit of *n* kbytes on the size of physical memory (BSD only).
−p	changes the pipe size to *n* (UNIX/RT only).
−s	imposes a size limit of *n* kbytes on the size of the stack area (BSD only).
−t	imposes a time limit of *n* seconds to be used by each process (BSD only).

If no option is given, **−f** is assumed. If *n* is not given the current limit is printed.

umask *nnn*
The user file-creation mask is set to *nnn* (see **umask**(2)). If *nnn* is omitted, the current value of the mask is printed.

unalias *name* ...
The parameters given by the list of *name*s are removed from the *alias* list.

unset −**f** *name* ...
The parameters given by the list of *name*s are unassigned, i. e., their values and attributes are erased. Readonly variables cannot be unset. If the flag, **−f**, is set, then the names refer to *function* names.

ver [*systype*[*command*]]
With no arguments, return the current value of the SYSTYPE environment variable that specifies the version of UNIX commands being executed by the shell. With a *systype* argument, change the SYSTYPE environment variable to either **bsd4.3** or **sys5.3**, depending on which is specified.

wait *n* Wait for the specified child process and report its termination status. If *n* is not given then all currently active child processes are waited for. The return code from this command is that of the process waited for.

whence −v *name* ...
For each *name*, indicate how it would be interpreted if used as a command name. The flag −v produces a more verbose report.

Invocation.

If the shell is invoked by exec(2), and the first character of argument zero (**$0**) is −, then the shell is assumed to be a *login* shell and commands are read from **/etc/profile** and then from either **.profile** in the current directory or **$HOME/.profile**, if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment parameter **ENV** if the file exists. If the −s flag is not present and *arg* is, then a path search is performed on the first *arg* to determine the name of the script to execute. The script *arg* must have read permission and any *setuid* and *getgid* settings will be ignored. Commands are then read as described below; the following flags are interpreted by the shell when it is invoked:

−c *string*	If the −c flag is present then commands are read from *string*.
−s	If the −s flag is present or if no arguments remain then commands are read from the standard input. Shell output, except for the output of the *Special commands* listed above, is written to file descriptor 2.
−i	If the −i flag is present or if the shell input and output are attached to a terminal (as told by *ioctl*(2)) then this shell is *interactive*. In this case TERM is ignored (so that **kill 0** does not kill an interactive shell) and INTR is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.
−r	If the −r flag is present the shell is a restricted shell.
−D*name*=*value*	You can use the −**D** option to specify a parameter *name* that will be set to *value* and then passed into the shell's environment. This SysV option is useful for tailoring the environment of a shell invoked from a program that is not another shell (such as the Display Manager). If the ENV parameter is given in this way, the startup script it specifies will be run. Note that any number of −**D** options can be specified.

The remaining flags and arguments are described under the **set** command above.

rksh Only

Note: SysV does not support rksh. **rksh** is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of **rksh** are identical to those of **ksh**, except that the following are disallowed:

changing directory (see **cd**(1)),
setting the value of **SHELL**, **ENV**, or **PATH**,
specifying path or command names containing /,
redirecting output (> and >>).

The restrictions above are enforced after **.profile** and the **ENV** files are interpreted.

When a command to be executed is found to be a shell procedure, **rksh** invokes **ksh** to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by **rksh** Some systems also provide a restricted editor **red**.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above). If the shell is being used non-interactively then execution of the shell file is abandoned. Runtime errors detected by the shell are reported by printing the command or function name and the error condition. If the line number that the error occurred on is greater than one, then the line number is also printed in square brackets ([]) after the command or function name.

CAVEATS

If a command which is a *tracked alias* is executed, and then a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to exec the original command. Use the **–t** option of the **alias** command to correct this situation.

Some very old shell scripts contain a ^ as a synonym for the pipe character | .

If a command is piped into a shell command, then all variables set in the shell command are lost when the command completes.

Using the **fc** built-in command within a compound command will cause the whole command to disappear from the history file.

The built-in command **.** *file* reads the whole file before any commands are executed. Therefore, **alias** and **unalias** commands in the file will not apply to any functions defined in the file.

FILES

/etc/passwd
/etc/profile
/etc/suid_profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cat(1), cd(1), echo(1), emacs(1), env(1), gmacs(1), newgrp(1), test(1), umask(1), vi(1), dup(2), exec(2), fork(2), ioctl(2), lseek(2), pipe(2), signal(2), umask(2), ulimit(2), wait(2), rand(3), a.out(5), profile(5), environ(7).

NAME

las – list objects mapped into the address space

SYNOPSIS

las [*options*]

DESCRIPTION

las produces a list of objects mapped into the address space. Information printed includes the virtual address range, the starting address within the object, and its pathname if available (in that order).

This command is most useful for system-level debugging.

OPTIONS

If no options are specified, **las** lists the address space of the current process.

–all	List all address space, including that occupied by Aegis.
–f[rom] *address*	Begin listing at the hexadecimal *address* specified.
–t[o] *address*	End listing at the hexadecimal *address* specified.

EXAMPLES

1.

$ las

```
      VA Range    Obj Start   Pathname

    8000 -    17FFF        0   /sys/node_data/global_data
   18000 -    2FFFF        0   /lib/pmlib
   30000 -    37FFF        0   /lib/syslib.peb
   38000 -    4FFFF        0   /lib/kslib
   50000 -    57FFF        0   /lib/trait_type_lib
   58000 -    67FFF    10000   /sys/node_data/global_data
   68000 -    9FFFF        0   /lib/streams
   A0000 -    A7FFF        0   /lib/vfmt_streams
   A8000 -    BFFFF        0   /lib/error
   C0000 -    E7FFF        0   /lib/swtlib
   E8000 -    F7FFF        0   /lib/ftnlib
   F8000 -    FFFFF        0   /lib/pbulib
  100000 -   127FFF        0   /lib/gprlib
  128000 -   14FFFF        0   /lib/clib
  150000 -   157FFF        0   /lib/lisp_initlib
  158000 -   15FFFF        0   /sys/node_data/global_rws
  160000 -   16FFFF    20000   /sys/node_data/global_data
  170000 -   187FFF        0   /lib/shlib
  188000 -   19FFFF        0   /lib/tfp
```

```
    1A0000 -   1BFFFF       0    /lib/dialoglib
    1C0000 -   1C7FFF       0    /sys/node_data/ipc_data
    1D0000 -   1D7FFF   30000    /sys/node_data/global_data
    200000 -   2AFFFF       0    -- temporary file --
    2B0000 -   2B7FFF       0    /sys/node_data/dm_mbx
    2B8000 -   2BFFFF       0    /com/sh
    2C0000 -   2C7FFF       0    -- temporary file --
    2C8000 -   2CFFFF       0    /com/las
    2D0000 -   2F7FFF   B0000    -- temporary file --
    BC0000 -   BCFFFF       0    /help_area/worksite
    BD0000 -   BDFFFF       0    /jtj

    2944 KB mapped.
```

2.

$ las –from 188000

```
         VA Range     Obj Start   Pathname

    188000 -   19FFFF       0    /lib/tfp
    1A0000 -   1BFFFF       0    /lib/dialoglib
    1C0000 -   1C7FFF       0    /sys/node_data/ipc_data
    1D0000 -   1D7FFF   30000    /sys/node_data/global_data
    200000 -   2AFFFF       0    -- temporary file --
    2B0000 -   2B7FFF       0    /sys/node_data/dm_mbx
    2B8000 -   2BFFFF       0    /com/sh
    2C0000 -   2C7FFF       0    -- temporary file --
    2C8000 -   2CFFFF       0    /com/las
    2D0000 -   2F7FFF   B0000    -- temporary file --
    BC0000 -   BCFFFF       0    /help_area/worksite
    BD0000 -   BDFFFF       0    /jtj

    1408 KB mapped.
```

3.

$ **las –f 188000 –t 200000**

```
      VA Range    Obj Start   Pathname

  188000 -   19FFFF        0   /lib/tfp
  1A0000 -   1BFFFF        0   /lib/dialoglib
  1C0000 -   1C7FFF        0   /sys/node_data/ipc_data
  1D0000 -   1D7FFF    30000   /sys/node_data/global_data

  288 KB mapped.
```

NAME

lbr2ar – convert lbr libraries to SR10 archive libraries

SYNOPSIS

lbr2ar [–y *dirname*] *lbrfile arfile*

DESCRIPTION

The **lbr2ar** command converts pre-SR10 lbr library files containing object modules in OBJ format to SR10 ar library archive files containing object modules in COFF format. The **lbr2ar** command extracts each object module from the *lbrfile*, executes the **obj2coff** converter to convert them to COFF, and creates a library archive (*arfile*) containing the converted object modules. Note that both the library format and the format of the individual object modules are changed.

OPTIONS

–y *dirname*	This option allows you to specify a new pathname, *dirname*, for the location of **obj2coff**. The new pathname for **obj2coff** is *dirname*/**obj2coff**. The default pathname for **obj2coff** is **/usr/apollo/bin**.

FILES

/usr/apollo/bin/obj2coff	**obj2coff** converter
/tmp/obj/*	Temporary files
/tmp/coff/*	Temporary files

SEE ALSO

obj2coff(1), ar(1).

NAME

lcm – load a color map

SYNOPSIS

lcm [–p *pathname*]

DESCRIPTION

lcm loads a color map from a file that specifies a set of color map entries. Each entry establishes an association between an index and a color value. When the DM is initially loaded, it sets the node's color map from the file in /sys/**dm/color_map**.

If no *pathname* is given, **lcm** loads the color map from /sys/**dm/color_map**. In this case, all 16 colors (that is, color entries for color slots 0-15) are reloaded. If you specify a *pathname*, **lcm** reads the given file and tries to load the colors associated with the indexes.

NOTE

If there are direct mode graphics programs running that have changed the color values for color slots 0-15, the execution changes the colors in these windows as well as resetting the DM's colors.

OPTIONS

–p *pathname*	Specify the file that contains the color values for red, green, and blue. The format of this file should be identical to the DM's color map file, /sys/**dm/color_map**. For more information about the format of this file, please refer to the manual *Programming with Domain Graphics Primitives*.

EXAMPLES

Load the DM's color map found in the file /sys/**dm/color_map**.

```
$ lcm
```

Load the color map specified in the file **my_colormap**.

```
$ lcm –p my_colormap
```

NAME

ld – link editor for common object files

SYNOPSIS

ld [*options*] *filename*

DESCRIPTION

ld combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and **ld** combines the objects, producing an object module that can either be executed or, if the **–r** *option* is specified, used as input for a subsequent **ld** run. **ld**'s output is left in **a.out**. By default this file is executable if no errors occurred during the load. If any input file (*filename*) is not an object file, **ld** assumes it is either an archive library or a text file containing link editor directives.

If any argument is a library, it is searched once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. The library (archive) symbol table (see **ar** (4)) is searched sequentially with as many passes as are necessary to resolve external references which can be satisfied by library members. Thus, the ordering of library members is functionally unimportant, unless there exist multiple library members defining the same external symbol.

OPTIONS

–a	Creates an absolute file. This is the default if the **–r** *option* is not used. Used with the **–r** *option*, **–a** allocates memory for common symbols.
–e *epsym*	Sets the default entry point address for the output file to be that of the symbol *epsym*.
–f *fill*	Sets the default fill pattern for "holes" within an output section as well as initialized *bss* sections. The argument *fill* is a two-byte constant.
–l*x*	Searches a library **lib***x***.a**, where *x* is up to nine characters. A library is searched when its name is encountered, so the placement of a **–l** is significant. By default, libraries are located in **LIBDIR** or **LLIBDIR**.
–L *dir*	Changes the algorithm of searching for **lib***x***.a** to look in *dir* before looking in **LIBDIR** and **LLIBDIR**. This *option* is effective only if it precedes the **–l** *option* on the command line.
–m	Produces a map or listing of the input/output sections on the standard output.
–M	Outputs a message for each multiply-defined external definition.

−o *outfile*	Produces an output object file by the name *outfile*. The name of the default object file is **a.out**.
−r	Retains relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent **ld** run. The link editor does not notice unresolved references, and the output file is not executable unless **−a** is also specified.
−s	Strips line number entries and symbol table information from the output object file.
−t	Turns off the warning about multiply-defined symbols that are not the same size.
−u *symname*	Enters *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. The placement of this *option* on the **ld** line is significant; it must be placed before the library which will define the symbol.
−V	Outputs a message giving information about the version of **ld** being used.
−VS *num*	Uses *num* as a decimal version stamp identifying the **a.out** file that is produced. The version stamp is stored in the optional header.
−x	Does not preserve local symbols in the output symbol table; enters external and static symbols only. This *option* saves some space in the output file.
−Y*[LU],dir*	Changes the default directory used for finding libraries. If **L** is specified, the first default directory **ld** searches, **LIBDIR**, is replaced by *dir*. If **U** is specified, and **ld** has been built with a second default directory, **LLIBDIR**, then that directory is replaced by *dir*. If **ld** was built with only one default directory and **U** is specified, a warning is printed and the *option* is ignored.
−T *systype*	Defines the target system type (systype) for the compiled object. Same as **−A sys[type]**,*sys*.

Domain/OS SysV EXTENSIONS

The Domain/OS SysV version of **ld**, includes support for features which are not available on System V. Domain **ld** includes support for the following extensions: Static Resource Information records (**.sri**), Module Information records (**.mir**), and control of global variable visibility.

Each of the following *options* must be preceded by the –A switch.

–A sys[type],*sys*
: Sets the environment variable SYSTYPE to *sys* while **ld** is running. If *sys* is any, SYSTYPE is not reset. A Static Resource Information (SRI) record for systype is produced. This *option* is useful to set the resolution of systype-dependent links; e.g., if your systype is **bsd4.3**, and you specify: –A systype,sys5.3, **/usr/lib/libm.a** resolves to /sys5.3/usr/lib/libm.a, instead of **/bsd4.3/usr/lib/libm.a**.

–A **run[type]**,*sys*
: Determines the system call semantics of the object module. For example, if you specify **-A** systype,any, and specify **-A runtype,bsd4.3**, the executable resolves pathnames according to your current SYSTYPE value, but always uses **bsd4.3** system call semantics.

–A stacksize,*hexnum*
: Produces a stacksize (SRI) with the specified value; *hexnum* is a one to eight digit hexadecimal number, optionally preceded by **0**x or **0X**.

–A exp[unge],*sym1*,*sym2*
: Removes the defined global symbol from the symbol table. No subsequent link runs (using **ld** or **bind**) will be able to resolve to this symbol. The symbol will not be entered into the KGT if this object is installed, nor will it be visible if this object is part of an archive.

–A **allexp[unge]** [–A **keep[sym]**,*sym1*,*sym2*]
: Expunges all defined symbols except those specified. If this *option* appears multiple times, all symbols specified will be kept in the symbol table.

–A **looks[ection]**,*sec1*,*sec2*
: Makes the named section available for sharing with a public section in an installed library.

–A **alllooks[ection]**
: Makes all sections available for sharing with their counterpart public sections in an installed library.

–A marks[ection],*sec1,sec2*
: Makes the specified section names (*sec1,sec2*) public. Affects only those object files that are destined to be installed as an installed library.

–A allmarks[ection]
: Makes all sections public. Affects only those object files that are destined to be installed as an installed library.

–A nolooks[ection],*sec1,sec2*
: Makes the named sections (*sec,sec2*) unavailable for sharing.

–A allnolooks[ection]
: Makes all data named sections unavailable for sharing.

–A nomarks[ection],*sec1,sec2*
: Makes the named sections private.

–A allnomarks[ection]
: Makes all sections private.

–A inlib,*pathname1,pathname2*
: Specifies the pathnames of the libraries to be installed at load time.

–A noinlib,*pathname1,pathname2*
: Deletes the named libraries from the list of libraries to be installed at load time.

–A loadhigh Creates an Static Resource Information (SRI) record to instruct the loader to load the object at the "high" end of memory for position-independent code.

–A module,*name*
: Specifies the object module name of the output binary file. This name will be stored in an object file Module Information Record (MIR). The default name of the object module is the name of the first object module read.

–A nosys[tem]
: Does not make system globals visible (turns off the check in the KGT (Known Global Table) and installed libraries).

–A allres[olved]
: Exits with an error status if unresolved references exist. This is the default.

–A noallres[olved]
: Terminates successfully even if unresolved references exist (providing there are no other errors during linking). This *option* is useful when running **ld** from shell scripts or other drivers, such as **/bin/cc**, or **/bin/make**.

CAVEATS

Through its *options* and input directives, the common link editor provides great flexibility; however, you must assume some added responsibilities. Input directives and *options* should ensure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, you must not place any object at virtual address zero in the program's address space.
- When the link editor is called through cc (1), a startup routine is linked with your program. This routine calls exit() after execution of the main program. If you call the link editor directly, you must ensure that the program always calls **exit()** rather than falling through the end of the entry routine (see **exit(2)**).

The symbols *etext*, *edata*, and *end* (see **end(3C)**) are reserved and are defined by the link editor. Your program may not redefine them.

If the link editor does not recognize an input file as an object file or an archive file, it assumes that it contains link editor directives and attempts to parse it. This occasionally produces an error message complaining about "syntax errors".

Arithmetic expressions can only have one forward–referenced symbol per expression.

FILES

LIBDIR/libx.a	Libraries
LLIBDIR/libx.a	Libraries
a.out	Output file
LIBDIR	Usually **/lib**
LLIBDIR	Usually **/usr/lib**

SEE ALSO

ar(1), cc(1), end(3), ar(5), a.out(5), systype(1M)

NAME

lex – generate programs for simple lexical tasks

SYNOPSIS

lex [**–rctvn**] [*file*] ...

DESCRIPTION

lex generates programs to be used in simple lexical analysis of text.

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

The file **lex.yy.c** is generated. When loaded with the library, this file copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in *yytext*, an external character array. Matching is done in order of the strings in the file. Strings can contain square brackets to indicate character classes, as in **[abx–z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators *****, **+**, and **?** mean respectively any non-negative number of, any positive number of, and either zero or one occurrence of, the previous character or character class. The character **.** is the class of all ASCII characters except newline. Parentheses for grouping and vertical bar for alternation are also supported.

The notation *r*{*d*,*e*} in a rule indicates between *d* and *e* instances of regular expression *r*. It has higher precedence than /, but lower than *, ?, +, and concatenation. Thus [a–zA–Z]+ matches a string of letters. The character ^ at the beginning of an expression permits a successful match only immediately after a newline, and the character $ at the end of an expression requires a trailing newline. The character / in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream. An operator character can be used as an ordinary symbol if it is used within double quotes ("), or preceded by a backslash (\).

Three subroutines defined as macros are expected: **input()** to read a character; **unput(***c***)** to replace a character read; and **output(***c***)** to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named **yylex()**, and the library contains a main function (**main()**) that calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function **yymore()** accumulates additional characters into the same *yytext*; and the function **yyless(***p***)** pushes back the portion of the string matched beginning at *p*, which should be between *yytext* and *yytext+yyleng*. The macros *input* and *output* use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes double percent characters (%%) it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a %%, as in YACC. Lines preceding %% which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with {}. Note that curly brackets do not imply parentheses; only string substitution is done.

External names generated by **lex** all begin with the prefix yy or YY.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p *n*	number of positions is *n* (default 2500)
%n *n*	number of states is *n* (500)
%e *n*	number of parse tree nodes is *n* (1000)
%a *n*	number of transitions is *n* (2000)
%k *n*	number of packed character classes is *n* (1000)
%o *n*	size of output array is *n* (3000)

The use of one or more of the above automatically implies the −v option, unless the −n option is used.

OPTIONS

−r	Indicates RATFOR actions.
−c	Indicates C actions. This is the default.
−t	Causes the **lex.yy.c** program to be written instead to standard output.
−v	Provides a one-line summary of statistics.
−n	Does not print out the −v summary.

Multiple files are treated as a single file. If no files are specified, standard input is used.

EXAMPLE

```
D       [0-9]
%%
if      printf("IF statement\n");
[a-z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"      skipcommnts();
%%

 skipcommnts()
 {
        for (;;)
        {
               while (input() != '*')
                      ;
               if (input() != '/')
                      unput(yytext[yyleng-1]);
               else
                      return;
        }
 }
```

BUGS

The **−r** option is not yet fully operational.

SEE ALSO

yacc(1).
Domain/OS Programming Environment Reference.

NAME

line – read one line

SYNOPSIS

line

DESCRIPTION

line copies one line (up to a newline) from the standard input and writes it on the standard output. It returns an exit code of 1 on **EOF** and always prints at least a newline. It is often used within shell files to read from the user's terminal.

SEE ALSO

sh(1).
read(2) in the *SysV Programmer's Reference*.

NAME

lint – a C program checker

SYNOPSIS

lint [*option*] ... *file* ...

DESCRIPTION

lint attempts to detect features of C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are:

- Unreachable statement
- Loops not entered at the top
- Automatic variables declared and not used
- Logical expressions whose value is constant
- Functions that return values in some places and not in others
- Functions called with varying numbers or types of arguments
- Functions whose values are not used or whose values are used but none returned.

Arguments whose names end with **.c** are taken to be C source files. Arguments whose names end with **.ln** are taken to be the result of an earlier invocation of **lint** with either the **–c** or the **–o** option used. The **.ln** files are analogous to **.o** (object) files that are produced by the cc(1) command when given a **.c** file as input. Files with other suffixes are warned about and ignored.

lint takes all the **.c**, **.ln**, and **llib-l***x***.ln** (specified by **–l***x*) files and processes them in their command line order. By default, **lint** appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. However, if the **–p** option is used, the portable C lint library (**llib-port.ln**) is appended instead. When the **–c** option is not used, the second pass of **lint** checks this list of files for mutual compatibility. When the **–c** option is used, the **.ln** and the **llib-l***x***.ln** files are ignored.

Any number of **lint** options can be used, in any order, intermixed with file-name arguments.

OPTIONS

Options to Suppress Complaints

–a Suppresses complaints about assignments of long values to variables that are not long.

–b Suppresses complaints about **break** statements that cannot be reached. (Programs produced by **lex** or **yacc** often result in many such complaints).

–h Does not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

–u Suppresses complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running **lint** on a subset of files of a larger program).

–v Suppresses complaints about unused arguments in functions.

–x Does not report variables referred to by external declarations but never used.

Options That Alter lint's Behavior

–l*x* Includes additional **lint** library **llib-l***x***.ln**. For example, you can include a **lint** version of the math library **llib-lm.ln** by inserting **–lm** on the command line. This argument does not suppress the default use of **llib-lc.ln**. These **lint** libraries must be in the assumed directory. This option can be used to reference local **lint** libraries and is useful in the development of multi-file projects.

–n Does not check compatibility against either the standard or the portable **lint** library.

–p Attempts to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

–c Causes **lint** to produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of **lint**'s first pass only, and are not checked for inter-function compatibility.

–o lib Causes **lint** to create a **lint** library with the name **llib-l***lib***.ln**. The **–c** option nullifies any use of the **–o** option. The **lint** library produced is the input given to **lint**'s second pass. The **–o** option simply causes this file to be saved in the named **lint** library. To produce a **llib-l***lib***.ln** without extraneous messages, use the –x option. The –v option is useful if the source file(s) for the **lint** library are just external interfaces (for example, the way the file **llib-lc** is written). These option settings are also available through the use of "lint comments" (see below).

The –D, –U, and –I options of **cpp**(1) and the –g and –O options of **cc**(1) are also recognized as separate arguments. The –g and –O options are ignored, but, by recognizing these options, **lint**'s behavior is closer to that of the cc(1) command. Other options are warned about and ignored. The pre-processor symbol "lint" is defined to allow certain questionable code to be altered or removed for **lint**. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by **lint**.

Certain conventional comments in the C source change the behavior of **lint**:

/*NOTREACHED*/
: At appropriate points stops comments about unreachable code. (This comment is typically placed just after calls to functions like **exit**(2)).

/*VARARGS*n**/
: Suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/
: Turns on the −v option for the next function.

/*LINTLIBRARY*/
: At the beginning of a file shuts off complaints about unused functions and function arguments in this file. This is equivalent to using the −v and −x options.

lint produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the −c option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name is printed followed by a question mark.

The behavior of the −c and the −o options allows for incremental use of **lint** on a set of C source files. Generally, one invokes **lint** once for each source file with the −c option. Each of these invocations produces a **.ln** file which corresponds to the **.c** file, and prints all messages that are about just that source file. After all the source files have been separately run through **lint**, it is invoked once more (without the −c option), listing all the **.ln** files with the needed −l*x* options. This prints all the inter-file inconsistencies. This scheme works well with **make**(1); it allows **make** to **lint** only the source files that have been modified since the last time the set of source files were **lint**ed.

BUGS

exit(2), **setjmp**(3C), and other functions that do not return are not understood; this causes various lies.

FILES

LLIBDIR	The directory where the lint libraries specified by the −l*x* option must exist, usually **/usr/lib**
LLIBDIR/lint[12]	First and second passes
LLIBDIR/llib-lc.ln	Declarations for C Library functions (binary format; source is in **LLIBDIR**/llib-lc)
LLIBDIR/llib-port.ln	Declarations for portable functions (binary format; source is in **LLIBDIR**/llib-port)
LLIBDIR/llib-lm.ln	Declarations for Math Library functions (binary format; source is in **LLIBDIR**/llib-lm)

TMPDIR/*lint*	Temporaries
TMPDIR	Usually /**usr**/**tmp** but can be redefined by setting the environment variable **TMPDIR** (see **tempnam**() in **tmpnam**(3S)).

SEE ALSO

cc(1), cpp(1), make(1).

NAME

list – produce C source listing from a common object file

SYNOPSIS

list [–V] [**–h**] [–F *function*] *source–file* . . . [*object–file*]

DESCRIPTION

list produces a C source listing with line number information attached. If multiple C source files were used to create the object file, **list** accepts multiple file names. The object file is taken to be the last non-C source file argument. If no object file is specified, the default object file, **a.out**, is used.

Line numbers are printed for each line marked as breakpoint inserted by the compiler (generally, each executable C statement that begins a new line of source). Line numbering begins again for each function. Line number 1 is always the line containing the left brace ({) that begins the function body. Line numbers are also supplied for inner block redeclarations of local variables so that they can be distinguished by the symbolic debugger.

OPTIONS

–V	Prints, on standard error, the version number of the **list** command executing.
–h	Suppresses heading output.
–F*function*	Lists only the named function. The –F option can be specified multiple times on the command line.

CAVEATS

Object files given to **list** must have been compiled with the –g option of **cc**(1).

Since **list** does not use the C preprocessor, it may be unable to recognize function definitions whose syntax has been distorted by the use of C preprocessor macro substitutions.

DIAGNOSTICS

list produces the error message ''list: name: cannot open'' if *name* cannot be read. If the source file names do not end in **.c** , the message is ''list: name: invalid C source name''. An invalid object file causes the message ''list: name: bad magic'' to be produced. If some or all of the symbolic debugging information is missing, one of the following messages is printed: ''list: name: symbols have been stripped, cannot proceed'', ''list: name: cannot read line numbers'', and ''list: name: not in symbol table''. The following messages are produced when *list* has become confused by **#ifdef**'s in the source file: ''list: name: cannot find function in symbol table'', ''list: name: out of sync: too many }'', and ''list: name: unexpected end-of-file''. The error message ''list: name: missing or inappropriate line numbers'' means that either symbol debugging information is missing, or *list* has been confused by C preprocessor statements.

SEE ALSO

cc(1), ld(1).

NAME

llib – list installed libraries

SYNOPSIS

llib [**–a**]

DESCRIPTION

The **llib** command lists those libraries which have been installed in the current process via the build-in **inlib** shell. These libraries are used to resolve unknown references when loading a program. To find out if a symbol is known and will be used in resolving an unknown reference, use esa.

OPTIONS

–a	Also list those libraries which are known globally to every process. These libraries are installed at boot time using the configuration information in /etc/sys.conf.

SEE ALSO

sh(1), csh(1), ksh(1)

NAME

llkob – list locked objects

SYNOPSIS

llkob [*options*]

DESCRIPTION

llkob lists the locked objects resident on volumes mounted on this node, and objects resident in other nodes that are locked by processes running locally.

The listing for each object includes the locking constraints imposed on the object (for example, n-readers XOR 1-writer), the specific lock mode being used (for example, read, write, read-intending-write), the network node ID of the node at which the object is located, the node ID of the node in which the locking process is active, and the name (if it is available) of the object itself.

OPTIONS

–r[emote]	Specify list of only those objects that either reside on this node and are locked by another node, or reside on another node and are locked by this node (that is, those objects whose locks are in some way remote).
–c[ount]	List only a one-line summary of the number of objects locked.

EXAMPLES

$ llkob

```
                        HOME   LOCKING
USE  CONSTRAINT         NODE    NODE        FILE

 W   nR_xor_1W           21      21         /sys/dm/pdb
 R   nR_xor_1W           21      21         /sys/dm/fonts/std
 W   nR_xor_1W           21      21         --Temporary File--
 R   nR_xor_1W           21      21         --Uncataloged Permanent File--
 W   nR_xor_1W           21      21         --Display Manager Pad--
```

$ llkob –c

```
 locked: 102 -- 100 local, 2 remote; 100 locally locked, 2 remotely
```

NAME

ln – create a hard or soft link

SYNOPSIS

ln *name* [*target*]
ln –s *name target*
ln *name* ... *directory*

DESCRIPTION

ln creates both hard and soft links. A link is a directory entry that refers to a file. You can have several links, in one or more directories, to the same file. Changes to a file are effective whether or not the file is referenced through a link.

A hard link is indistinguishable from the original directory entry. Hard links may not span file systems and may not refer to directories.

A soft (or symbolic) link contains a pathname. Symbolic links may span file systems and may refer to directories.

An **open**(2) operation on a link opens the referenced file. A **stat**(2) on a soft link is equivalent to a **stat** on the file that the link points to. Use **lstat**(2) to obtain information about the link itself. The **readlink**(2) call is useful for reading the contents of a soft link.

Given one or two arguments, **ln** creates a link to an existing file *name*. If *target* is given, the link has that name. The *target* argument may also be a directory in which to place the link. If *target* is not a directory, the link is placed in the current directory.

When the –s option is used, **ln** requires that a *target* be specified. If *target* exists, **ln** –s will fail. If only the directory is specified, the link is made to the last component of *name*.

Given more than two arguments, **ln** makes links to all the named files in the named directory. The links made will have the same name as the files being linked to.

OPTIONS

–f	Forces creation of the link if permitted by access modes (hard links only).
–s	Creates soft (symbolic) links.

NOTE

By default, **ln** generates a hard link.

SEE ALSO

cp(1), mv(1), rm(1), link(2), readlink(2), lstat(2), stat(2).

NAME

logger – make entries in the system log

SYNOPSIS

logger [**–t** *tag*] [**–p** *pri*] [**–i**] [**–f** *file*] [*message* ...]

DESCRIPTION

logger provides a program interface to the **syslog**(3) system log module.

You can give **logger** a message on the command line, which is logged immediately, or **logger** can read a file and log each line.

OPTIONS

–t *tag*	Mark every line in the log with the specified *tag*.
–p *pri*	Enter the message with the specified priority. You can specify the priority numerically or as a ''facility.level'' pair. For example, **–p local3.info** logs the message(s) as *info*rmational level in the *local3* facility. The default is ''user.notice.''
–i	Log the process ID of the logger process with each line.
–f *file*	Log the specified *file*.
message	The message to log; if you do not specify one, **logger** logs the **–f** file or standard input.

EXAMPLES

```
$ logger System rebooted

$ logger –p local0.notice –t HOSTIDM –f /dev/idmc
```

SEE ALSO

syslog(3), syslogd(1M)

NAME

login – sign on

SYNOPSIS

login [**–p**] [*username*] [*env-var* ...]

DESCRIPTION

The **login** command is used when a user initially signs on, or it may be used at any time to change from one user to another. The latter case is the one summarized above and described here. See *Getting Started with Domain/OS* for information on initially logging in.

If **login** is invoked without an argument, it prompts you for a username, and, if appropriate, a password. Echoing is turned off (if possible) while you type the password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that requires you to enter a second ''dialup'' password. This only occurs for dial-up connections, and is prompted by the message ''dialup password:''. Both passwords are required for a successful login.

After a successful login, accounting files are updated and the operating environment is set from the **~/.environ** file if it exists, or from **/etc/environ** if **~/.environ** doesn't exist.

If your environment is BSD, you are informed of the existence of mail (see **mail**(1)). For all environments, the message of the day (**/etc/motd**) is printed. Both are suppressed if you have a **.hushlogin** file in your home directory; this is mostly used to make life easier for non-human users, such as **uucp**(1C).

The **login** command initializes the user and group IDs and the working directory, and modifies the environment as follows (see **environ**(7)).

The basic SysV environment is initialized to:

HOME=*your-log-in-directory*
LOGNAME=*your-log-in-name*
MAIL=**/usr/mail/***your-log-in-name*
NODEID=*your-node's-hexadecimal-id*
NODETYPE=*your-node's-model-number*
ORGANIZATION=*your-organization-name*
PATH=**:/bin:/usr/bin:/usr/apollo/bin**
PROJECT=*your-project-name*
SHELL=*last-field-of-passwd-entry*
SYSTYPE=sys5.3
TERM=*your-terminal-type*
TZ=*timezone-specification*
USER=*your-log-in-name*

The basic BSD environment is initialized to:

> HOME=*your-log-in-directory*
> LOGNAME=*your-log-in-name*
> MAIL=**/usr/spool/mail/***your-log-in-name*
> NODEID=*your-node's-hexadecimal-id*
> NODETYPE=*your-node's-model-number*
> ORGANIZATION=*your-organization-name*
> PATH=**:/usr/ucb:/bin:/usr/bin:/usr/apollo/bin**
> PROJECT=*your-project-name*
> SHELL=*last-field-of-passwd-entry*
> SYSTYPE=**bsd4.3**
> TERM=*your-terminal-type*
> TZ=*timezone-specification*
> USER=*your-log-in-name*

The **–p** argument causes the remainder of the environment to be preserved, otherwise any previous environment is discarded.

The environment can be expanded or modified by supplying additional arguments to **login**, either at execution time or when **login** requests your log-in name. Arguments can take either the form *xxx* or *xxx*=*yyy*. Arguments without an equal sign are placed in the environment as

> **L***n*=*xxx*

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing = are placed in the environment without modification. If they already appear in the environment, then they replace the older value, with two exceptions. The variables PATH and SHELL cannot be changed. Both **login** and **getty** understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

After setting up the environment, **login** executes a command interpreter (for example, a shell) as specified in the last field of your **/etc/passwd** file entry. If this field in /etc/passwd is empty, the default command interpreter is **/bin/sh** for the BSD and SysV environments, and **/com/sh** for the Aegis environment. See **csh**(1), **ksh**(1), and **sh**(1) for a description of the shell's startup behavior. Argument 0 of the command interpreter is the name of the command interpreter with a leading dash (''–'').

NOTES

If the file /etc/**nologin** exists, **login** prints the contents of this file on your terminal and exits. This is used by **shutdown**(8) to stop you from logging in when the system is about to go down.

login is recognized by **sh**(1) and **csh**(1) and executed directly (without forking).

An undocumented option, **–r** is used by the remote **login** server, **rlogind**(1M) to force **login** to enter into an initial connection protocol. **–h is used by telnetd**(1M) and other servers to list the host from which the connection was received.

SECURITY

Sites wishing additional security protection on dial-up lines may want to use these security features, /etc/d_users and /etc/d_passwd. /etc/d_users is simply a file containing a list of users authorized to log in on this node.

/etc/d_passwd is a file containing lines of the following format:

/bin/sh:*encrypted-password*

where *encrypted-password* is the dial-in password for the specified shell as returned by **crypt**(3). If an entry for the user's log-in shell is not found in this file, the password for **/bin/sh** is used.

FILES

/etc/utmp	Accounting
/etc/wtmp	Accounting
/usr/mail/*your-name*	Mailbox for user *your-name*
/etc/motd	Message of the day
/etc/passwd	Password file
.hushlogin	Makes **login** quieter

DIAGNOSTICS

"Login incorrect," Username or password cannot be matched.

"No Shell", "cannot open password file", "no directory": consult a UNIX system programming counselor.

BUGS

login(1) is not currently used for logging into the Display Manager or the Server Process Manager, although the procedure used by those programs is similar.

The System V Release 3 facility for using a "*" in the shell field of the **/etc/passwd** file is not supported by Domain/OS.

SEE ALSO

mail(1), newgrp(1), sh(1), su(1M).
passwd(4), profile(4), environ(5) in the *SysV Programmer's Reference*.

NAME

logname – get login name

SYNOPSIS

logname

DESCRIPTION

logname returns the contents of the environment variable **$LOGNAME**, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1).
logname(3X), environ(5) in the *SysV Programmer's Reference*.

NAME

lorder – find ordering relation for an object library

SYNOPSIS

lorder *file* . . .

DESCRIPTION

Input is one or more object or library archive *files* (see **ar**(1)). Standard output is a list of pairs of object file or archive member names, meaning that the first file of the pair refers to external identifiers defined in the second. Output may be processed by **tsort**(1) to find an ordering of a library suitable for one-pass access by **ld**(1). Note that the link editor **ld**(1) is capable of multiple passes over an archive in the portable archive format [see **ar**(4)] and does not require that **lorder**(1) be used when building an archive. Using **lorder**(1) may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing **.o** files.

```
ar –cr library `lorder *.o  tsort`
```

FILES

TMPDIR/*symref	Temporary files
TMPDIR/*symdef	Temporary files

TMPDIR is usually **/usr/tmp** but can be redefined by setting the environment variable **TMPDIR** (see **tempnam()** in **tmpnam(3S)**).

CAVEAT

lorder accepts as input any object or archive file, regardless of its suffix, provided there is more than one input file. If there is but a single input file, its suffix must be **.o**.

SEE ALSO

ar(1), ld(1), tsort(1), ar(4).

NAME

lp, **cancel** – send/cancel requests to an LP line printer

SYNOPSIS

lp [**–c**] [**–d***dest*] [**–m**] [**–n***number*] [**–o***option*] [**–s**] [**–t***title*] [**–w***files*]
cancel [ids] [printers]

DESCRIPTION

lp arranges for the named files and associated information (collectively called a "request") to be printed by a line printer. If no file names are mentioned, the standard input is assumed. A dash (–) used as a file name indicates the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

lp associates a unique "id" with each request and prints it on the standard output. This id can be used later to cancel (see **cancel**) or find the status (see **lpstat**(1)) of the request.

OPTIONS

The following options to **lp** may appear in any order and may be intermixed with file names:

–c Makes copies of the *file(s)* to be printed immediately when **lp** is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the –c option is not given, then you should be careful not to remove any of the *file(s)* before the request has been printed completely. Without the –c option, any changes made to the named *files* after the request is made, but before it is printed, will be reflected in the printed output.

–d*dest* Chooses *dest* as the printer or class of printers where printing will take place. If *dest* is a printer, the request will be printed only on that specific printer. If *dest* is a class of printers, the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer availability, file space limitation, etc.), requests for specific destinations may not be accepted (see **accept**(1M) and **lpstat**(1)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see **lpstat**(1)).

–m Sends mail after the *files* have been printed (see **mail**(1)). By default, no mail is sent.

–n*number* Prints *number* copies of the output (default is 1).

–o*option* Specifies a printer-dependent or class-dependent *option*. Several such *options* may be collected by specifying –o more than once. For more information about what are valid *options*, see Models in **lpadmin**(1M).

–s	Suppresses messages from **lp**(1) such as "request id is ...".
–t*title*	Prints *title* on the banner page of the output.
–w	Writes a message to your terminal after the *files* have been printed. If you are not logged in, mail is sent instead.

Cancel cancels line printer requests made by **lp**(1). The command line arguments can be either request *id*s (as returned by **lp**(1)) or *printer* names (for a complete list of *printer* names, use **lpstat**(1)). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which is currently printing on that *printer*. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1).
accept(1M), lpadmin(1M), lpsched(1M) in the *Managing SysV System Software*.

NAME

lpstat – print LP status information

SYNOPSIS

lpstat [*options*]

DESCRIPTION

lpstat prints information about the current status of the LP spooling system.

If no *options* are given, **lpstat** prints the status of all requests made to **lp**(1). Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). **lpstat** prints the status of such requests. *Options* can appear in any order and may be repeated and intermixed with other arguments. Some *options* can be followed by an optional *list* that can be in one of two forms:

- A list of items separated from one another by a comma
- A list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

  lpstat –u"user1, user2, user3"

The omission of a *list* following these *options* causes all information relevant to the *option* to be printed, for example:

lpstat –o

prints the status of all output requests.

OPTIONS

–a[*list*]	Prints acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.
–c[*list*]	Prints class names and their members. *List* is a list of class names.
–d	Prints the system default destination for *lp*.
–o[*list*]	Prints the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.
–p[*list*]	Prints the status of printers. *List* is a list of printer names.
–r	Prints the status of the LP request scheduler
–s	Prints a status summary, including the system default destination, a list of class names and their members, and a list of printers and their associated devices.
–t	Prints all status information.
–u[*list*]	Prints status of output requests for users. *List* is a list of login names.
–v[*list*]	Prints the names of printers and the path names of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1).

NAME

ls – list contents of directory

SYNOPSIS

ls [–RSadCxmlnogrtucpFbqisfT] [*names*]

DESCRIPTION

For each directory argument, **ls** lists the contents of the directory. For each file argument, **ls** repeats its name and any other information requested. By default, it sorts the output alphabetically. If you specify no argument, **ls** lists the current directory. If you give several arguments, **ls** first sorts the arguments appropriately, but prints file arguments before directories and their contents.

ls produces lists in three major formats. By default, it lists one entry per line. It can also generate a multi-column format, as well as stream output format in which files are listed across the page, separated by commas.

OPTIONS

–R Recursively list subdirectories encountered. This *option* follows soft links unless the –S *option* is also used.

–S Shows link text rather than the object to which the link has been made.

–a List all entries. Usually, entries whose names begin with a period (.) are not listed.

–d If an argument is a directory, list only its name (not its contents). Often used with –l to get the status of a directory.

–C Produce multi-column output with entries sorted down the columns.

–x Produce multi-column output with entries sorted across rather than down the page. (–S is ignored.)

–m Produce stream output format.

–l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file. If the file is a special file, the size field contains the major and minor device numbers, rather than a size. The mode printed under the –l *option* consists of 10 characters, interpreted as follows:

The first character is:

d	Directory
b	Block special file
c	Character special file
p	Fifo (also called a ''named pipe'') special file
–	Ordinary file

The next nine characters are interpreted as three sets of three bits each.

The first set refers to the owner's permissions; the next to permissions of others in the user group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, execute permission is interpreted as permission to search the directory for a specified file.

The permissions are indicated as follows:

r	File is readable
w	File is writable
x	File is executable
–	Indicated permission is *not* Granted
l	Mandatory locking will occur during access (the set-group-ID bit is on and the group execution bit is off)
s	The set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on
S	Undefined bit-state (the set-user-ID bit is on and the user execution bit is off)
t	The 1000 (octal) bit, or sticky bit, is on (see **chmod**(1)), and execution is on
T	The 1000 bit is turned on, and execution is off (undefined bit-state)

For user and group permissions, the third position (the execute permission) is sometimes occupied by a character other than x or –. An s or S may occupy this position, indicating the state of the set-user-ID or set-group-ID bit. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as root but need to assume the identity of the user stated at "login." The group execute permission may be given as **l**, indicating that mandatory file and record locking will occur. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access. For others permissions, the third position may be occupied by **t** or T, indicating the state of the sticky bit and execution permissions.

–n Same as **–l**, except that the owner's UID and group's GID numbers are printed, rather than the associated character strings.

–o Same as **–l**, except that the group is not printed.

–g Same as **–l**, except that the owner is not printed.

–r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

–t Sort by time modified (latest first) instead of by name.

–u Use time of last access instead of last modification for sorting (with the **–t** *option*) or printing (with the **–l** *option*).

–c Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**–t**) or printing (**–l**).

–p Put a slash (/) after each filename if that file is a directory.

–F Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable. If used with –S, softlinks will appear with an @ symbol.

–b Force nongraphic characters to be printed in the octal \ddd notation.

–q Force nongraphic characters in filenames to be printed as question marks (?).

–i For each file, print the i-number in the first column of the report.

–s Give size in blocks, including indirect blocks, for each entry. Block size is considered to be 1024. Print a total count of blocks when the sizes of the files in a directory are listed.

–f Force each argument to be interpreted as a directory and list the name found in each slot. Turn off **–l**, **–t**, **–s**, and **–r**, and turn on **–a**; the order is the order in which entries appear in the directory.

–T Used with the **-l** *option*, **–T** shows the "Apollo type" of each file.

EXAMPLES

The first set of examples refers to permissions.

To describe a file that is readable, writable, and executable by the user and readable by the group and others:

```
-rwxr--r--
```

To describe a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it:

```
-rwsr-xr-x
```

To describe a file that is readable and writable only by the user and the group and can be locked during access:

```
-rw-rwl---
```

The following examples describe the output from **ls**.

ls –l

(the long list) prints its output as follows:

```
-rwxrwxrwx  1 smith  dev    10876  May 16 9:42 part2
```

Reading from right to left, you see that the current directory holds one file, named "part2". Next, the last time that file's contents were modified was 9:42 A.M. on May 16. The file is moderately sized, containing 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group "dev" (perhaps indicating development), and his or her login name is "smith." The number, in this case "1," indicates the number of links to file "part2". Finally, the row of letters beginning with a dash (–) tells you that user, group, and others have permissions to read, write, execute "part2." The execute (x) symbol here occupies the third position of the three-character sequence. A – in the third position would have indicated a denial of execution permissions.

ls -aisn

prints its output as follows:

```
923765600  16 -rwxrwxrwx  1 20123  38    16329  Sep 3 14:34
```

This command lists all entries (a), including those whose names begin with a period (.). The output shows the **i**-number (the memory address of the i-node associated with the file) printed in the left-hand column (**i**); the size (in blocks) of the files, printed in the column to the right of the i-numbers (s); finally, the report is displayed in the numeric version of the long list, printing the **UID** (instead of user name) and **GID** (instead of group name) numbers associated with the files.

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls –l** command leads you to believe.

BUGS

Unprintable characters in filenames may confuse the columnar output *options*.

FILES

/etc/passwd	To get user IDs for **ls –l** and **ls –o**
/etc/group	To get group IDs for **ls –l** and **ls –g**
/usr/lib/terminfo/?/*	To get terminal information

SEE ALSO

chmod (1), find (1).

NAME

lsacl – list access control list

SYNOPSIS

lsacl [–odfsvmnlaLR] *file* . . .

DESCRIPTION

If you do not specify an *option*, **lsacl** shows the access control list (ACL) associated with the files and directories specified. It lists entries one per line in the following format:

%.%.% prwxk

where *%.%.%* represents a subject identifier (SID) in person.group.organization form, and **prwxk** represents a set of rights. If one of the letters **prwxk** appears, the associated right is present, if a – appears, it isn't. See **acl**(7) for more information about access rights. If you specify more than one file, the ACL is preceded by the filename.

OPTIONS

–o Lists the ACL associated with the specified objects. This is default if you do not specify an *option.*

–d Shows how new sub-directories created in the specified directory will inherit their protections. (This output is also known as the initial directory ACL.)

–f Shows how new files created in the specified directory will inherit their protections. (This output is also known as the initial file ACL.)

–s Lists any sub-system information. Protected sub-systems are an Aegis analogue to setuid programs, and you should use the commands available in the Aegis environment to manipulate them. This *option* is provided so that UNIX users can be aware of files that use the feature.

–v Selects verbose output. In this mode, each ACL (object, initial file and initial directory) is preceded by a label.

–m Shows the rights mask for extended ACL entries. By default, this information is not shown.

–n Shows the node/network access rights.

–l Shows a long listing, equivalent to selecting all the above *options.*

–a Shows all bits in the rights field, rather than showing [ignore] and [umask]. See **chacl**(1) for a description of these bits.

–L Directs **lsacl** to follow any soft links encountered, and operate on the object to which the link points. Since soft links in Domain/OS do not have ACLs, attempting to change or display the ACL on a link without the –L flag simply results in a warning.

−R Recursively lists the ACLs of any directories specified on the command line.

Only directories have additional fields relating to inheritance of protections for new sub-directories and files. If you specify a non-directory with either the **−d** or −f *option*, **lsacl** ignores them for that object.

SEE ALSO

chacl(1), cpacl(1), dbacl(1), chmod(1), chgrp(1), chown(1), ls(1), salacl(1M) umask(2) acl(5)

NAME

lty – list installed types

SYNOPSIS

lty [*options*]

DESCRIPTION

lty lists the types currently installed on a volume. It can also be used to list the contents of internal caches for debugging purposes.

OPTIONS

If no options are specified, **lty** lists types installed on the boot volume.

–n *node_spec*	Specify the node whose type names are to be listed. You may also specify the entry directory of a volume mounted for software installation, as shown in the example below.
–u	Display type UIDs as well as type names.
–glob	Display contents of global type name cache instead of the type file (for debugging only).
–priv	Display the contents of the private (per-user) type name cache instead of the type file (for debugging only).

EXAMPLES

```
$ lty
Local type file

area   bitmap  boot   casehm  ddf   evetype  hdru    ipad
lheap  mbx     mt     nil     null  obj      objlib  pad
pipe   rec     sch    sio     uasc  und
```

In the following example, the disk has been mounted for software installation. The disk's top level directory (cataloged as **/mounted_disk** by the **mount**(1M) command) must contain a sys directory. If it does not, you get a "types file not found" error.

```
$ /etc/mount /mounted_disk
$ lty –n /mounted_disk
Type file for "//my_node/mounted_disk"

area   bitmap  boot  casehm  ddf   evetype  hdru    ipad
lheap  mbx     mt    nil     null  obj      objlib  pad
pipe   rec     sch   sio     uasc  und
```

SEE ALSO

crty(1), dlty(1), inty(1), mount(1M)

NAME

m4 – macro processor

SYNOPSIS

m4 [*options*] [*files*]

DESCRIPTION

m4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is –, the standard input is read. The processed text is written on the standard output.

OPTIONS

–e	Operates interactively. Interrupts are ignored and the output is unbuffered.
–s	Enables line sync output for the C preprocessor (#line . . .)
–B*int*	Changes the size of the push-back and argument collection buffers from the default of 4,096.
–H*int*	Changes the size of the symbol table hash array from the default of 199. The size should be prime.
–S*int*	Changes the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.
–T*int*	Changes the size of the token buffer from the default of 512 bytes.

To be effective, these must appear before any file names and before **–D** or **–U**:

–D*name* [=*val*]
    Defines *name* to *val* or to null in *val*'s absence.

–U*name*
    Undefines *name*.

Macro calls have the form:

    name(arg1,arg2, . . ., argn)

The left parenthesis (must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

BUILT-IN MACROS

m4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define	Installs the second argument as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; $# is replaced by the number of arguments; $* is replaced by a list of all the arguments separated by commas; $@ is like $*, but each argument is quoted (with the current quotes).
undefine	Removes the definition of the macro named in its argument.
defn	Returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.
pushdef	Like *define*, but saves any previous definition.
popdef	Removes current definition of its argument(s), exposing the previous one, if any.
ifdef	If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on UNIX system versions of *m4*.
shift	Returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.
changequote	Changes quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., \` ´).
changecom	Changes left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.
divert	**m4** maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.
undivert	Causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another

	diversion. Undiverting discards the diverted text.
divnum	Returns the value of the current output stream.
dnl	Reads and discards characters up to and including the next new-line.
ifelse	Contains three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.
incr	Returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
decr	Returns the value of its argument decremented by 1.
eval	Evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
len	Returns the number of characters in its argument.
index	Returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.
substr	Returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
translit	Transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	Returns the contents of the file named in the argument.
sinclude	Is identical to *include*, except that it says nothing if the file is inaccessible.
syscmd	Executes the UNIX system command given in the first argument. No value is returned.
sysval	Is the return code from the last call to *syscmd*.
maketemp	Fills in a string of XXXXX in its argument with the current process ID.
m4exit	causes immediate exit from **m4**. Argument 1, if given, is the exit code the default is 0.

m4wrap	Argument 1 is pushed back at final EOF; example: m4wrap(`cleanup()')
errprint	prints its argument on the diagnostic output file.
dumpdef	Prints current names and definitions, for the named items, or for all if no arguments are given.
traceon	With no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
traceoff	Turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

SEE ALSO

cc(1), cpp(1).

NAME

mail, **rmail** – send mail to users or read mail

SYNOPSIS

Sending mail:

mail [–oswt] *persons*

rmail [–oswt] *persons*

Reading mail:

mail [–ehpqr] [–f *file*] [–F *persons*]

DESCRIPTION

Sending mail:

A *person* is usually a user name recognized by **login**(1). When *persons* are named, **mail** assumes a message is being sent (except in the case of the –F option). It reads from the standard input up to an end-of-file (CTRL/D), or until it reads a line consisting of just a period. When either of those signals is received, **mail** adds the *letter* to the *mailfile* for each *person*. A *letter* is a *message* preceded by a *postmark*. The message is preceded by the sender's name and a *postmark*. A *postmark* consists of one or more 'From' lines followed by a blank line (unless the –s argument was used).

If a letter is found to be undeliverable, it is returned to the sender with diagnostics that indicate the location and nature of the failure. If **mail** is interrupted during input, the file **dead.letter** is saved to allow editing and resending. The **dead.letter** file is recreated every time it is needed, erasing any previous contents.

The **rmail** command only permits the sending of mail; **uucp**(1C) uses **rmail** as a security precaution.

If the local system has the Basic Networking Utilities installed, mail may be sent to a recipient on a remote system. Prefix *person* by the system name and exclamation point. A series of system names separated by exclamation points can be used to direct a letter through an extended network.

Reading Mail:

The **mail** program, unless otherwise influenced by command-line arguments, prints a user's mail messages in last-in, first-out order. For each message, the user is prompted with a question mark (?), and a line is read from the standard input. The following commands are available to determine the disposition of the message:

<newline>, +, or **n**	Go on to next message.
d, or **dp**	Delete message and go on to next message.
d #	Delete message number #. Do not go on to next message.
dq	Delete message and quit **mail**.

h	Display a window of headers around current message.
h #	Display header of message number #.
h a	Display headers of all messages in the user's *mailfile*.
h d	Display headers of messages scheduled for deletion.
p	Print current message again.
–	Print previous message.
a	Print message that arrived during the *mail* session.
#	Print message number #.
r [*users*]	Reply to the sender, and other *user(s)*, then delete the message.
s [*files*]	Save message in the named *files* (**mbox** is default).
y	Same as save.
u [#]	Undelete message number # (default is last read).
w [*files*]	Save message, without its top-most header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons*.
q, or **ctl-d**	Put undeleted mail back in the *mailfile* and quit **mail**.
x	Put all mail back in the *mailfile* unchanged and exit **mail**.
!*command*	Escape to the shell to do *command*.
?	Print a command summary.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using **mail**.

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. A "Forwarded by..." message is added to the header. This is especially useful in a multi-machine environment to forward all of a person's mail to a single machine, and to keep the recipient informed if the mail has been forwarded. Installation and removal of forwarding is done with the **–F** option.

To forward all of one's mail to **systema!user** enter the following:

mail –Fsystema!user

To forward to more than one user, enter this command line:

mail –F"user1,systema!user2,systema!systemb!user3"

Note that when more than one user is specified, the whole list should be enclosed in double quotes so that it may all be interpreted as the operand of the –F option. The list can be up to 1024 bytes; either commas or white space can be used to separate users.

To remove forwarding, enter the following:

mail –F ""

The pair of double quotes is mandatory to set a NULL argument for the –F option.

In order for forwarding to work properly, the *mailfile* should have "mail" as group ID and the group permission should be read-write.

OPTIONS

Sending mail:

–o Suppresses the address optimization facility.

–s Suppresses the addition of a <newline> at the top of the letter being sent. See WARNINGS below.

–w Causes a letter to be sent to a remote user without waiting for the completion of the remote transfer program.

–t Causes a **To:** line to be added to the letter, showing the intended recipients.

Reading mail:

–e Causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.

–h Causes a window of headers to be displayed rather than the latest message. The display is followed by the '?' prompt.

–p Causes all messages to be printed without prompting for disposition.

–q Causes **mail** to terminate after interrupts. Normally an interrupt causes only the termination of the message being printed.

–r Causes messages to be printed in first-in, first-out order.

–f*file* Causes BI mail to use *file* (e.g., **mbox**) instead of the default *mailfile*.

–F*persons*
Entered into an empty *mailbox*, causes all incoming mail to be forwarded to *persons*.

WARNING

The "Forward to person" feature may result in a loop, if **sys1!userb** forwards to **sys2!userb** and **sys2!userb** forwards to **sys1!userb**. The symptom is a message saying "unbounded...saved mail in dead.letter."

The −s option should be used with caution. It allows the text of a message to be interpreted as part of the postmark of the letter, possibly causing confusion to other **mail** programs. To allow compatibility with **mailx**(1), if the first line of the message is "Subject:...", the addition of a <newline> is suppressed whether or not the −s option is used.

BUGS

Conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*user*	incoming mail for *user*; i.e., the **mailfile**
$HOME/mbox	saved mail
$MAIL	variable containing path name of **mailfile**
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), mailx(1), write(1).
Managing SysV System Software.

NAME

mailx – interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The command **mailx** provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, **mailx** provides commands to facilitate saving, deleting, and responding to messages. When sending mail, **mailx** allows editing, reviewing and other modification of the message as it is entered.

Many of the remote features of **mailx** will only work if the Basic Networking Utilities are installed on your system.

Incoming mail is stored in a standard file for each user, called the *mailbox* for that user. When **mailx** is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the **mbox** and is normally located in the user's HOME directory (see "MBOX" (ENVIRONMENT VARIABLES) for a description of this file). Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the **–f** option of the **mailx** command. Messages in the secondary file can then be read or otherwise processed using the same COMMANDS as in the primary *mailbox*. This gives rise within these pages to the notion of a current *mailbox*.

On the command line, *options* start with a dash (–) and any other arguments are taken to be destinations (recipients). If no recipients are specified, **mailx** will attempt to read messages from the *mailbox*.

When reading mail, **mailx** is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating **mailx** can accept regular commands (see COMMANDS below). When sending mail, **mailx** is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, **mailx** will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of **mailx** is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the **set** and **unset** commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If mail is found to to undeliverable, an attempt is made to return it to the sender's *mailbox*. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as **lp**(1) for recording outgoing mail on paper. Alias groups are set by the alias command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

> [**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, print is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many commands take an optional list of messages (*msglist*) to operate on. The default for *msglist* is the current message. A *msglist* is a list of message identifiers separated by spaces, which may include:

n	Message number **n**.
.	The current message.
^	The first undeleted message.
$	The last message.
*****	All messages.
n–m	An inclusive range of message numbers.
user	All messages from **user**.
/string	All messages with **string** in the subject line (case ignored).
:*c*	All messages of type *c*, where *c* is one of:

d	Deleted messages
n	New messages
o	Old messages
r	Read messages
u	Unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions (see *sh*(1)). Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, **mailx** tries to execute commands from the optional system-wide file (**/usr/lib/mailx/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. With the exceptions noted below, regular commands are legal inside start-up files. The most common use of a start-up file is to set up initial display options and alias lists. The following commands are not legal in the start-up file: !, Copy, edit, followup, Followup, **hold**, mail, **preserve**, reply, Reply, **shell**, and visual. An error in the start-up file causes the remaining lines in the file to be ignored. The **.mailrc** file is optional, and must be constructed locally.

OPTIONS

–e	Test for presence of mail. The **mailx** command prints nothing and exits with a successful return code if there is mail to read.
–f [*filename*]	Read messages from *filename* instead of *mailbox*. If no *filename* is specified, the **mbox** is used.
–F	Record the message in a file named after the first recipient. Overrides the "record" variable, if set (see ENVIRONMENT VARIABLES).
–h *number*	The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops. (See **addsopt** under ENVIRONMENT VARIABLES)
–H	Print header summary only.
–i	Ignore interrupts. See also "ignore" (ENVIRONMENT VARIABLES).
–n	Do not initialize from the system default *mailx.rc* file.
–N	Do not print initial header summary.
–r *address*	Pass *address* to network delivery software. All tilde commands are disabled. (See **addsopt** under ENVIRONMENT VARIABLES)
–s *subject*	Set the Subject header field to *subject*.
–u *user*	Read *user*'s *mailbox*. This is only effective if *user*'s *mailbox* is not read protected.
–U	Convert **uucp**-style addresses to internet standards. Overrides the "conv" environment variable. (See **addsopt** under ENVIRONMENT VARIABLES)

COMMANDS

The following is a complete list of **mailx** commands:

	Escape to the shell. See "SHELL" (ENVIRONMENT VARIABLES).
# *comment*	Null command (comment). This may be useful in *.mailrc* files.
=	Print the current message number.

?

Prints a summary of commands.

group *alias name ...*

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, alternates prints the current list of alternate names. See also "allnet" (ENVIRONMENT VARIABLES).

cd [*directory*]

chdir [*directory*]

Change directory. If *directory* is not specified, $HOME is used.

copy [*msglist*] *filename*

Copy messages to the file without marking the messages as saved. Otherwise equivalent to the save command.

Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the Save command.

Delete messages from the *mailbox*. If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES).

ignore [*header-field* ...]

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The Print and Type commands override this command.

dp [*msglist*]

dt [*msglist*]

Delete the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a delete command followed by a print command.

echo *string* ...

Echo the given strings (like echo(1)).

edit [*msglist*]

Edit the given messages. The messages are placed in a temporary file and the "EDITOR" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES). Default editor is ed(1).

exit

xit

Exit from **mailx**, without changing the *mailbox*. No messages are saved in the **mbox** (see also **quit**).

file [*filename*]

folder [*filename*]

Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

%	Current *mailbox*.
%user	*mailbox* for *user*.
#	Previous file.
&	Current **mbox**.

Default file is the current *mailbox*.

folders

Print the names of the files in the directory set by the "folder" variable (see ENVIRONMENT VARIABLES).

Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the "record" variable, if set. See also the Followup, Save, and Copy commands and "outfolder" (ENVIRONMENT VARIABLES).

Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the followup, Save, and Copy commands and "outfolder" (ENVIRONMENT VARIABLES).

from [*msglist*]

Prints the header summary for the specified messages.

alias *alias name ...*

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

Prints the page of headers which includes the message specified. The "screen" variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the z command.

help

Prints0 a summary of commands.

hold [*msglist*]

preserve [*msglist*]

Holds the specified messages in the *mailbox*.

if *s* | *r*

mail-commands

else

mail-commands

endif

Conditional execution, where *s* will execute following *mail-commands*, up to an **else** or **endif**, if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the *.mailrc* file.

discard *header-field ...*

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The Print and Type commands override this command.

list

Prints all commands available. No explanation is given.

mail *name ...*

Mail a message to the specified users.

Mail *name*

Mail a message to the specified user and record a copy of it in a file named after that user.

Arrange for the given messages to end up in the standard **mbox** save file when **mailx** terminates normally. See "MBOX" (ENVIRONMENT VARIABLES) for a description of this file. See also the exit and quit commands.

Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

| [*msglist*] [*shell-command*]

Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

hold [*msglist*]
: Preserve the specified messages in the *mailbox*.

Type [*msglist*]
: Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ignore** command.

print [*msglist*]

type [*msglist*]
: Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is **pg**(1) (see ENVIRONMENT VARIABLES).

quit
: Exit from **mailx**, storing messages that were read in **mbox** and unread messages in the *mailbox*. Messages that have been explicitly saved in a file are deleted.

Respond [*msglist*]
: Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

respond [*message*]
: Reply to the specified message, including all other recipients of the message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

Save [*msglist*]
: Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the Copy, followup, and Followup commands and "outfolder" (ENVIRONMENT VARIABLES).

save [*msglist*] *filename*
: Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when **mailx** terminates unless "keepsave" is set (see also ENVIRONMENT VARIABLES and the exit and quit commands).

set

set *name*

set *name=string*

set *name=number*

Define a variable called *name*. The variable may be given a null, string, or numeric value. Set by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the **mailx** variables.

shell

Invoke an interactive shell (see also "SHELL" (ENVIRONMENT VARIABLES)).

size [*msglist*]

Print the size in characters of the specified messages.

Read commands from the given file and return to command mode.

top [*msglist*]

Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the **mbox** upon normal termination. See exit and quit.

Print [*msglist*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the ignore command.

type [*msglist*]

print [*msglist*]

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is pg(1) (see ENVIRONMENT VARIABLES).

Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

unset *name* ...

Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version

Prints the current version and release date.

Edit the given messages with a screen editor. The messages are placed in a temporary file and the "VISUAL" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the save command.

xit

exit

Exit from **mailx**, without changing the *mailbox*. No messages are saved in the **mbox** (see also quit).

z[+ | −]

Scroll the header display forward or backward one screen–full. The number of headers displayed is set by the "screen" variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (˜). See "escape" (ENVIRONMENT VARIABLES) for changing this special character.

˜! *shell-command*
: Escape to the shell.

˜.
: Simulate end of file (terminate message input).

˜: *mail-command*
: Perform the command-level request. Valid only when sending a message while reading mail.

˜_ *mail-command*
: Perform the command-level request. Valid only when sending a message while reading mail.

˜?
: Print a summary of tilde escapes.

˜A
: Insert the autograph string "Sign" into the message (see ENVIRONMENT VARIABLES).

˜a
: Insert the autograph string "sign" into the message (see ENVIRONMENT VARIABLES).

~b *name ...*
Add the *names* to the blind carbon copy (Bcc) list.

~c *name ...*
Add the *names* to the carbon copy (Cc) list.

~d
Read in the *dead.letter* file. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~e
Invoke the editor on the partial message. See also "EDITOR" (ENVIRONMENT VARIABLES).

~f [*msglist*]
Forward the specified messages. The messages are inserted into the message, without alteration.

~h
Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~i *string*
Insert the value of the named variable into the text of the message. For example, ~A is equivalent to '~i Sign.'

~m [*msglist*]
Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~p
Print the message being entered.

~q
Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in *dead.letter*. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~r *filename*
Read in the specified file.

~< *filename*
Read in the specified file.

~< !*shell-command*

Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

~s *string ...*

Set the subject line to *string*.

~t *name ...*

Add the given *names* to the To list.

~v

Invoke a preferred screen editor on the partial message. See also "VISUAL" (ENVIRONMENT VARIABLES).

~w *filename*

Write the partial message onto the given file, without the header.

~x

Exit as with ~q except the message is not saved in *dead.letter*.

~| *shell-command*

Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within **mailx**.

HOME=*directory*

The user's base of operations.

MAILRC=*filename*

The name of the start-up file. Default is $HOME/.mailrc.

The following variables are internal **mailx** variables. They may be imported from the execution environment or set via the set command at any time. The **unset** command may be used to erase variables.

addsopt

Enabled by default. If */bin/mail* is not being used as the deliverer, **noaddsopt** should be specified. (See WARNINGS below)

allnet

All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet**. See also the **alternates** command and the "metoo" variable.

append

Upon termination, append messages to the end of the **mbox** file instead of prepending them. Default is **noappend.**

askcc

Prompt for the Cc list after message is entered. Default is **noaskcc**.

asksub

Prompt for subject if it is not specified on the command line with the –s option. Enabled by default.

autoprint

Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint**.

bang

Enable the special-casing of exclamation points (!) in shell escape command lines as in **vi**(1). Default is **nobang**.

cmd=*shell-command*

Set the default command for the **pipe** command. No default value.

conv=*conversion*

Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also "sendmail" and the **–U** command line option.

crt=*number*

Pipe messages having more than *number* lines through the command specified by the value of the "PAGER" variable (**pg**(1) by default). Disabled by default.

DEAD=*filename*

The name of the file in which to save partial letters in case of untimely interrupt. Default is $HOME/dead.letter.

debug

Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

dot

Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot**.

EDITOR=*shell-command*

The command to run when the edit or ~e command is used. Default is **ed**(1).

escape=*c*

Substitute *c* for the ~ escape character. Takes effect with next message sent.

folder=*directory*

The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a **mailx** command line, "folder" must be an exported *sh* environment variable. There is no default for the "folder" variable. See also "outfolder" below.

header

Enable printing of the header summary when entering **mailx**. Enabled by default.

hold

Preserve all messages that are read in the *mailbox* instead of putting them in the standard **mbox** save file. Default is **nohold**.

ignore

Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof

Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the **~.** command. Default is **noignoreeof**. See also "dot" above.

keep

When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave
Keep messages that have been saved in other files in the *mailbox* instead of deleting them. Default is **nokeepsave**.

MBOX=*filename*
The name of the file to save messages which have been read. The xit command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

metoo
If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

LISTER=*shell-command*
The command (and options) to use when listing the contents of the "folder" directory. The default is **ls**(1).

onehop
When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder
Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the path name is absolute. Default is **nooutfolder**. See "folder" above and the Save, Copy, followup, and Followup commands.

page
Used with the **pipe** command to insert a form feed after each message sent through the pipe. Default is **nopage**.

PAGER=*shell-command*
The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is **pg**(1).

prompt=*string*
Set the *command mode* prompt to *string*. Default is "? ".

quiet
Refrain from printing the opening message and version when entering **mailx**. Default is **noquiet**.

record=*filename*
Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save
Enable saving of messages in *dead.letter* on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen=*number*
Sets the number of lines in a screen–full of headers for the **headers** command.

sendmail=*shell-command*
Alternate command for delivering messages. Default is **mail**(1).

sendwait
Wait for background mailer to finish before returning. Default is **nosendwait**.

SHELL=*shell-command*
The name of a preferred command interpreter. Default is **sh**(1).

showto
When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*string*
The variable inserted into the text of a message when the ˜a (autograph) command is given. No default (see also ˜**i** (TILDE ESCAPES)).

Sign=*string*
The variable inserted into the text of a message when the ˜A command is given. No default (see also ˜**i** (TILDE ESCAPES)).

toplines=*number*
The number of lines of header to print with the **top** command. Default is 5.

VISUAL=*shell-command*
The name of a preferred screen editor. Default is **vi**(1).

FILES

$HOME/.mailrc	Personal start-up file
$HOM/mbox	Secondary storage file
/usr/mail/*	Post office directory
/usr/lib/mailx/mailx.help*	Help message files
/usr/lib/mailx/mailx.rc	Optional global start-up file
/tmp/R[emqsx]*	Temporary files

WARNINGS

The **−h**, **−r** and **−U** options can be used only if **mailx** is built with a delivery program other than */bin/mail*.

BUGS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

The full internet addressing is not fully supported by **mailx**. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a "." are treated as the end of the message by **mail**(1) (the standard mail delivery program).

SEE ALSO

ls(1), mail(1), pg(1).

NAME

make – maintain, update, and regenerate groups of programs

SYNOPSIS

make [–f makefile] [**–p**] [**–i**] [**–k**] [**–s**] [**–r**] [**–n**] [**–b**] [**–e**] [**–u**] [**–t**] [**–q**] [*names*]

DESCRIPTION

make allows the programmer to maintain, update, and regenerate groups of computer programs.

make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **–f** option is present, **makefile**, **Makefile**, and the Source Code Control System (SCCS) files, **s.makefile**, and **s.Makefile** are tried in order. If you use a dash (–) in place of *makefile*, the standard input is taken. More than one – *makefile* argument pair may appear.

Make updates a target only if its dependents are newer than the target (unless the **–u** option is used to force an unconditional update). All prerequisite files of a target are added recursively to the list of targets. Missing files are considered out-of-date.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a colon (:), then a (possibly null) list of prerequisite files or dependencies. Text following a semicolon (;) and all following lines that begin with a tab are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or a pound sign (#) begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash><newline> sequence. Everything printed by **make** (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

produces

```
ab
```

like the shell does.

Sharp (#) and newline surround comments.

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
cc a.o b.o –o pgm
a.o: incl.h a.c
cc –c a.c
b.o: incl.h b.c
cc –c b.c
```

Command lines are executed one at a time, each by its own shell. The **SHELL** environment variable can be used to specify which shell **make** should use to execute commands. The default is **/bin/sh**. The first one or two characters in a command can be the following: –, @, –@, or @–. If @ is present, printing of the command is suppressed. If – is present, **make** ignores an error. A line is printed when it is executed unless the **–s** option is present, or the entry **.SILENT:** is in *makefile*, or unless the initial character sequence contains a @. The **–n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see discussion of the **MAKEFLAGS** macro under *Environment*). The **–t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate **make**. If the **–i** option is present, or the entry **.IGNORE:** appears in *makefile*, or the initial character sequence of the command contains –. the error is ignored. If the **–k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **–b** option allows old makefiles (those written for the old version of **make**) to run without errors.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name **.PRECIOUS**.

OPTIONS

–f *makefile*	Description file name. *makefile* is assumed to be the name of a description file.
–p	Prints out the complete set of macro definitions and target descriptions.
–i	Ignorse error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.
–k	Abandons work on the current entry if it fails, but continues on other branches that do not depend on that entry.
–s	Silent mode. Does not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.
–r	Does not use the built-in rules.
–n	No execute mode. Prints commands, but does not execute them. Even lines beginning with an @ are printed.
–b	Compatibility mode for old makefiles.
–e	Environment variables override assignments within makefiles.
–u	Forces an unconditional update.
–t	Touches the target files (causing them to be up-to-date) rather than issue the usual commands.

−q	Question. The **make** command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

SPECIAL NAMES

.DEFAULT	If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.
.PRECIOUS	Dependents of this target will not be removed when quit or interrupt are hit.
.SILENT	Same effect as the −s option.
.IGNORE	Same effect as the −**i** option.

ENVIRONMENT

Make reads the environment, assuming all variables to be macro definitions and processing them as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The −**e** option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built-in rules.

The **MAKEFLAGS** environment variable is processed by **make** as containing any legal input option (except −**f** and −**p**) defined for the command line. Upon invocation, **make** ''invents'' the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for ''super-makes.'' In fact, as noted above, when the −**n** option is used, the command $(**MAKE**) is executed anyway; hence, one can perform a **make** −**n** recursively on a whole software system to see what would have been executed. This is because the −**n** is put in **MAKEFLAGS** and passed to further invocations of $(**MAKE**). This is one way of debugging all of the makefiles for a software project without actually doing anything.

INCLUDE FILES

If the string **include** appears as the first seven letters of a line in a *makefile,* and is followed by a blank or a tab, the rest of the line is assumed to be a file name and will be read by the current invocation, after substituting for any macros.

MACROS

Entries of the form *string1* = *string2* are macro definitions. *String2* is defined as all characters up to a comment character or an unescaped newline. Subsequent appearances of $(*string1*[:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, newline characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

INTERNAL MACROS

There are five internally maintained macros which are useful for writing rules for building targets.

$* The macro $* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@ The $@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$< The $< macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module which is out-of-date with respect to the target (i.e., the ''manufactured'' dependent file name). Thus, in the **.c.o** rule, the $< macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
        .c.o:
                cc –c –O $*.c
```

or:

```
        .c.o:
                cc –c –O $<
```

$? The $? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules which must be rebuilt.

$% The $% macro is only evaluated when the target is an archive library member of the form *lib(file.o)*. In this case, $@ evaluates to *lib* and $% evaluates to the library member, *file.o*.

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros, the meaning is changed to ''directory part'' for **D** and ''file part'' for **F**. Thus, $(@D) refers to the directory part of the string $@. If there is no directory part, ./ is generated. The only macro excluded from this alternative form is $?.

PRESET VARIABLES

The currently defined preset variables are:

```
ACC = /com/cc
ACFLAGS = –opt
AS = as
ASFLAGS =
BIND = /com/bind
BINDFLAGS =
CC = /bin/cc
CFLAGS = –O
F77FLAGS =
FFLAGS = –opt
```

```
FTN = ftn
GET = get
GFLAGS =
LD = /bin/ld
LDFLAGS =
LEX = lex
LFLAGS =
MAKE = make
MAKEFLAGS = b
PAS = /com/pas
PFLAGS = –opt
RF = rf
SHELL = /bin/sh
YACC = yacc
YFLAGS =
$ = $
```

SUFFIXES

Certain names (for instance, those ending with .o) have inferable prerequisites such as .c, .s, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, **make** has inference rules that allow files to be built from other files by examining the suffixes and determining an appropriate inference rule to use. The current default single suffix rules are:

```
.sh:
        cp $< $@; chmod 0777 $@

.f
        &(F77) $(F77FLAGS) $(LDFLAGS) $< -o $*

.pas:
        $(PAS) $@ $(PFLAGS)
        mvf $@.bin $@ -r

.c:
        $(CC) $(CFLAGS) $(LDFLAGS) $*.c -o $*
```

The current default double suffix rules are:

```
markfile.o: markfile
        A=@;echo "static char _sccsid[] = \042`grep $$A'(#)' markfile`\042;"\
                > markfile.c
        $(CC) markfile
        rm -f markfile.c

.c.a:
        $(CC) –c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm –f $*.o

.l.c:
        $(LEX) $(LfLAGS) $<
        mv lex.yy.c $@

.y.c:
        $(YACC) $(YFLAGS) $<
        mv y.tab.c $@

.l.o:
        $(LEX) $(LFLAGS) $<
        $(CC) $(CFLAGS) -c lex.yy.c
        rm lex.yy.c
        mv lex.yy.o $@

.y.o:
        $(YACC) $(YFLAGS) $<
        $(CC) $(CFLAGS) -c y.tab.c
        rm y.tab.c
        mv y.tab.o $@

.s.o:
        $(AS) $(ASFLAGS) -o $@ $<

.c.o:
        $(CC) $(CFLAGS) -c $<
```

The internal rules for **make** are contained in the source file **rules.c** for the **make** program. These rules can be locally modified. To print the rules compiled into the **make** on any machine in a form suitable for recompilation, use the following command:

# make –fp – 2>/dev/null </dev/null

A tilde in the above rules refers to an SCCS file see sccsfile(4) for more information about these files. Thus, the rule **.c˜.o** transforms an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with **make**'s suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., **.c:**) is the definition of how to build *x* from *x*.**c**. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

.SUFFIXES: .o .c .c˜ .y .y˜ .pas .l .l˜ .s .s˜ .sh .sh˜ .h .h˜ .f .f˜

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

INFERENCE RULES

The first example can be done more briefly.

```
pgm:  a.o  b.o
        cc  a.o  b.o  –o  pgm
a.o  b.o:  incl.h
```

This is because **make** has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to cc(1), lex(1), and yacc(1), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

LIBRARIES

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus

lib(file.o) and **$(LIB)(file.o)** both refer to an archive library which contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal.

Rules pertaining to archive libraries have the form *.XX*.**a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o)  lib(file2.o)  lib(file3.o)
        @echo  lib  is  now  up-to-date

.c.a:
        $(CC)  -c  $(CFLAGS)  $<
        $(AR)  $(ARFLAGS)  $@  $*.o
        rm  -f  $*.o
```

In fact, the **.c.a** rule listed above is built into **make** and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o)  lib(file2.o)  lib(file3.o)
        $(CC)  -c  $(CFLAGS)  $(?:.o=.c)
        $(AR)  $(ARFLAGS)  lib  $?
        rm  $?
        @echo  lib  is  now  up-to-date

.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files are out-of-date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

NOTES

Some commands return non-zero status inappropriately; use **–i** to overcome the difficulty.

BUGS

Filenames with the equal sign (=), colon (:), or at sign (@) characters will not work. Commands that are directly executed by the shell, notably **cd**(1), are ineffectual across newlines in **make**. The syntax *(lib(file1.o file2.o file3.o)* is illegal. You cannot build *lib(file.o)* from *file.o*. The macro $(a:.o=.c˜) does not work. Named pipes are not handled well.

FILES

[Mm]akefile and **s.[Mm]akefile**
/bin/sh

SEE ALSO

cc(1), lex(1), yacc(1), printf(3S), sccsfile(4), cd(1), sh(1).

NAME

man – print entries in this manual

SYNOPSIS

man [*options*] [*section*] *title(s)*

DESCRIPTION

man locates and prints the manual page(s) specified by the *title* argument. If you also specify a *section,* **man** looks for the *title* only in the *section* indicated (otherwise, it searches the whole manual). You must enter the *title* in lowercase, and the *section* number may not have a letter suffix.

man examines the environment variable $TERM and attempts to select options that adapt the output to the terminal being used. Refer to **environ**(5) for more information. The –T*term* option overrides the value of $TERM.

Section may be changed before each *title*.

OPTIONS

–T*term*	Prints the entry as appropriate for terminal type *term*. For a list of recognized values of *term*, type **help term2**. Also refer to **term**(5) for further details. The default value of *term* is 450.
–w	Prints on the standard output only the *pathnames* of the entries, relative to /usr/catman, or to the current directory for **–d** option.
–d	Searches the current directory rather than */usr/catman*. This option requires that you specify the full filename (e.g., **cu.1c**, rather than just **cu**).
–c	Invoke **col**(1). This is done automatically, unless *term* is one of the following: 300, 300s, 450, 37, 4000a, 382, 4014, tek, 1620, and X.

EXAMPLE

To reproduce this page on the terminal, as well as any other entries named **man** that may exist in other sections of the manual, use:

```
# man man
```

BUGS

man prints manual entries that were formatted by **nroff**(1) when the UNIX system was installed. Entries are originally formatted with terminal type 37, and are printed using the correct terminal filters as derived from the –T*term* and $TERM settings.

FILES

/usr/catman/?_man/man[1-8]/*
Preformatted manual entries; system and command reference directories

SEE ALSO

col(1)
term(5)
"

NAME

mcs – manipulate the object file comment section

SYNOPSIS

mcs [*options*] *object-file* ...

DESCRIPTION

The mcs command manipulates the comment section, normally the ''.comment'' section, in an object file. It is used to add to, delete, print, and compress the contents of the comment section in a UNIX System object file. The mcs command must be given one or more of the options described below. It takes each of the options given and applies them in order to the *object-files*. If the object file is an archive, the file is treated as a set of individual object files. For example, if the –a option is specified, the string is appended to the comment section of each archive element.

OPTIONS

–a *string*	Appends *string* to the comment section of the *object-files*. I *string* contains embedded blanks, it must be enclosed in quota tion marks.
–c	Compresses the contents of the comment section. All duplicat entries are removed. The ordering of the remaining entries is no disturbed.
–d	Deletes the contents of the comment section from the object file The object file comment section header is removed also.
–n *name*	Specifies the name of the section to access. By default, mc deals with the section named *.comment*. This option can be use to specify another section.
–p	Prints the contents of the comment section on the standard out put. If more than one name is specified, each entry printed i tagged by the name of the file from which it was extracted, usin the *filename*:*string* format.

EXAMPLES

To print *file*'s *comment* section:

mcs –p *file*

To append *string* to *file*'s *comment* section:

mcs –a *string file*

NOTE

mcs cannot add new sections or delete existing sections to executable objects wit magic number 0413 [see a.out(4)].

FILES

TMPDIR/mcs*
: Temporary files

TMPDIR/*
: Temporary files

TMPDIR is usually /usr/tmp but can be redefined by setting the environment variable **TMPDIR** [see **tmpnam**(3S)].

SEE ALSO

cpp(1), a.out(4).

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [**–n**] [**–y**]

DESCRIPTION

The command, with argument **n**, forbids messages via **write**(1) by revoking non-user write permission on the user's terminal. The **mesg** command, with argument **y**, reinstates permission. All by itself, **mesg** reports the current state without changing it.

FILES

/dev/tty*

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

SEE ALSO

write(1).

NAME

mkapr – make an Apollo product report

SYNOPSIS

mkapr [–v]

DESCRIPTION

The **mkapr** command creates a product report. This command replaces the **crucr** (create a user change request form) utility available in prior software releases.

Output from **mkapr** may be in either (or both) of two forms:

1. Printed, human-readable copy; or
2. Encoded, transmittable form.

Printed product reports should be sent to:

APR Administrator/Customer Services
M/S CHG 01 CS
Apollo Computer Inc.
330 Billerica Road
Chelmsford, MA 01824

Encoded product reports may be sent to Apollo Customer Services via the UUCP network. The network address is: **apollo!apr_cs_admin**
Recommended paths to Apollo are via **attunix**, **mit-eddie**, or **decwrd!decvax** (these paths may change). Customer Services will acknowledge all product reports received. Do not assume your product report has been received until you receive a reply. Security-conscious sites should not send confidential material. Voluminous submissions should be sent via magnetic media.

OPTIONS

mkapr supports only one option, –v. This will assert verbose mode; any system services called by by **mkapr** will be allowed to send output to the standard output and/or standard error devices. Normal mode operation is for **mkapr** to invoke the system services silently.

SERVICES SUPPORTED

In addition to creating Apollo Problem Reports online, **mkapr** will make available viewing, editing, printing and mailing services if they exist (and **mkapr** can find them). The mailing service known to **mkapr** is:

UNIX environment - **sendmail**

The print services known to **mkapr** are:

Aegis environment - **prf**
BSD environment - **lpr**
SysV environment - **lp**

If a desired service is not available to **mkapr**, a product report (print or send) file will be saved in the current directory for printing or sending at a later time.

DIALOG INTERFACE

mkapr will make use of the DIALOG graphic interface environment of the Apollo Domain system whenever possible. This interface is designed for ease of learning and use.

COMMAND DRIVEN INTERFACE

If the display environment you are using does not support the graphic interface, you will see the following prompt:

```
mkapr>
```

Entering the command 'help' will display the available commands. Here is the list of commands for reference:

Command	Description
help [mkapr]	List Commands. To display the help file, use the **mkapr** option.
change	Change APR Information Fields.
edit	Edit the detailed Problem Description.
view	View the current APR.
print	Print the current APR.
send	Send the current APR.
exit	Save current customer information changes (if any) and exit.
cancel	Exit without saving customer information changes.

You need only enter as much of any command as is necessary to uniquely identify it. For example, you need only type **ch** for the change command.

Detailed descriptions of commands

change Allow user to provide the necessary information prior to submitting an APR. There are 2 kinds of input here. First, information that is extracted from the system the user is on. Second, information that the user must input. Most field defaults (including system extractable data) will be overridable by the user. The date field is the only non-overridable field. A file exists between sessions which currently stores customer contact, name, address, and telephone information. This file is created upon the first invocation of the **mkapr** tool, is stored in the current working directory and is called **.aprinit**. Upon subsequent invocations of the **mkapr** tool, the customer information is used as the default for these fields.

Within the change command, the prompt becomes **mkapr..change>** Current input is then displayed by field. The user is asked to enter the field # to change, then asked to enter the changed value (entering <RETURN> effectively will abort the current change field # request leaving the field unchanged). The cycle is then repeated. Replying 'h' or 'help' at this point will display the following help message for the change command:

Change Command	Description
help [mkapr]	List commands. To display the help file, use the **mkapr** option.
display fields	Display all fields and their respective values.
change field *n*	Request to change the value of field # *n*. Pressing the RETURN key at the prompt `enter new value ==>` will leave the value unchanged.
exit	Exit the change command.

edit	An appropriate editor will be invoked according to available system services. The user should enter a detailed problem description and save and exit the editor in the appropriate manner. You will then be returned to the **mkapr>** prompt.
view	The current **mkapr** information will be displayed to the user in an appropriate manner according to available system services.
print	The current **mkapr** information will be printed to the default printer according to available system services.
send	The current **mkapr** information will be sent to Apollo Computer in an appropriate manner according to available system services.
exit	If any changes to customer information occurred during this session, save all customer information to the non-system
cancel	Exit **mkapr**. Do not save changes to customer information from this session.

INITIAL FIELD VALUES

The fields of an Apollo Problem Report that are collectively known as customer information Fields are initialized from a file read when **mkapr** starts up. These fields contain such information as the name of the customer contact, the name (company name) of the customer, and the customer's address and telephone number. The initialization file has the name **.aprinit** and the mkapr program will search for it. The search order for the initialization file is:

1. Look in the current working directory
2. Look in the home directory as given by the shell variable HOME
3. Look in the system directory /etc/apr

It is not an error for no initialization file to exist; mkapr will leave the customer information fields blank. The fields can be edited and the initialization file will be updated when **mkapr** exits.

The file /etc/apr/**.aprinit** is a special case; **mkapr** will not write to this file. The system administrator (or other privileged account) must create the directory /etc/**apr** with appropriate access permissions, then run **mkapr** to create a local copy of the file **.aprinit** and copy or move the file to the directory.

The initialization file is an ASCII text file that may be created and modified using any of the text editors available to you. The body of the **.aprinit** file created by **mkapr** is reproduced here:

```
# Comment lines begin with '#'
# Non-comment lines have the following form:
#   FIELD_NAME : FIELD_VALUE : IGNORED
# The field name must not be changed.
# The ':' character delimits fields.
# The field value may be changed; it must not contain ':'.
# unless the field value is quoted by either ' ' or " " pairs.
# Anything after the second ':' is thrown away.
#
customer_contact : A. Random User : 14
customer_name : Apollo Computer, Inc. : 21
customer_addr1 : CHF 02 RD : 9
customer_addr2 : 330 Billerica Road : 18
customer_addr3 : Chelmsford, MA  01824 : 21
customer_addr4 : USA : 3
customer_phone : 1-508-256-6600 x7739 : 20
mail_path : 'apollo!apr_cs_admin:' : 22
#
```

NOTES

Since **mkapr** assumes that the site mail facility (probably **sendmail**) knows how to get from your site to Apollo, you must edit the mail_path field value in **.aprinit** to give **mkapr** the correct path. Be sure that your mail facility is setup correctly. See your site administrator for help.

Run /usr/ucb/**newaliases** at least once before attempting to use **mkapr**'s send function.

Offsite mailing may not be allowed by your site. If so, you must make other arrangements to get mail to Apollo. See your site administrator for help.

FILES

/usr/apollo/bin/mkapr	The executable object
/usr/man/cat1/mkapr.1	This manual page (UNIX)
/sys/help/mkapr.hlp	This help file (AEGIS)
	Initial field values (search order):
.aprinit	(1st) (updated)
$HOME/.aprinit	(2nd) (updated)
/etc/apr/.aprinit	(last) (read only)
/tmp/apr.*	Temporary files:
apr.*.v	Product report view file
apr.*.p	Product report print file
apr.*.s	Product report send file
apr.*.c	Product report send command file
apr.*.e	Problem description edit file

NAME

mkdir – make directories

SYNOPSIS

mkdir [**–m** *mode*] [**–p**] *dirname* . . .

DESCRIPTION

mkdir creates specified directories in mode 777 (all access permissions granted). It automatically makes standard entries of dot (.) and dot-dot (..) for its parent.

The owner ID and group ID of the new directories are set to the process's real user ID and group ID, respectively.

OPTIONS

–m *mode*	Allows you to specify the *mode* to be used for new directories. Choices for *modes* can be found in **chmod**(1).
–p *dirname*	Creates *dirname* by creating all the non-existing parent directories first.

EXAMPLE

To create the subdirectory structure **ltr/jd/jan**, type the following:

```
mkdir –p ltr/jd/jan
```

BUGS

mkdir requires write permission in the parent directory.

umask(1) may alter the *mode* of specified directories normally created with mode 777.

DIAGNOSTICS

The **mkdir** command returns exit code 0 if all directories given in the command line were successfully made; otherwise, it prints a diagnostic and returns non-zero. An error code is stored in *errno*.

SEE ALSO

sh (1), rm (1), umask (1), intro (2), mkdir(2).

NAME

mksinit – create initialization code for STREAMS drivers and modules

SYNOPSIS

mksinit [–C *number* ...] *master_file* ...

DESCRIPTION

mksinit is a tool to create the Apollo-implementation-dependent initialization code for STREAMS drivers and modules. The input to **mksinit** is a set of **master** files. The output of **mksinit** is an **init.c** file in the current directory, which must be compiled and linked with the module(s) and/or driver(s). The entry point of the module must be the **mksinit** routine in **init.c**. The following is a sample module build:

```
mksinit master.d/sample*
/bin/cc –c init.c
/bin/ld –e mksinit init.o other module objects
```

OPTIONS

–C *numbers* Provides the number of controllers for the **master** files processed. Each number corresponds in the order specified to one of the **master** files.

SEE ALSO

master(4)

NAME

mmt, mvt – typeset documents, viewgraphs, and slides

SYNOPSIS

mmt [*options*] [*files*]

DESCRIPTION

Although very similar to **mm**(1), these two commands typeset their input via **troff**(1), as opposed to formatting it via **nroff**(1). The **mmt** command uses the MM macro package for its operations. The **mvt** command uses the Macro Package for View Graphs and Slides. Preprocessing through **tbl**(1) and **eqn**(1) is available for both. The proper pipelines and the required arguments and flags for **troff**(1) and the appropriate macro packages are generated, depending on the options selected.

Arguments or flags other than those given below are passed to **troff**(1) or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* argument. If no arguments are given, **mmt** prints a list of its options.

OPTIONS

–e	Invokes **eqn**(1) and causes it to read the */usr/pub/eqnchar* file. See **eqnchar**(5) for details concerning this file.
–t	Invokes **tbl**(1).
–T*dest*	Creates output for **troff**(1) device *dest*.
–Di10	Directs the output to the local Imagen Imprint-10 laser printer.
–a	Invokes the **–a** option of **troff**(1).
–y	Uses the noncompacted version of the macros. This is the default.
–z	Invokes no output filter to process or redirect the output of **troff**(1).

BUGS

If you specify simply a dash (–), **–e** option, and/or **–t** option along with the *–olist* option of **troff**(1), a harmless ''broken pipe'' diagnostic may result. This usually happens under these conditions if the last page of the document is not specified in *list*.

DIAGNOSTICS

m[mv]t: no input file

None of the arguments is a readable file and the command is not used as a filter.

SEE ALSO

env (1), eqn (1), mm (1), nroff (1), tbl (1), troff (1), environ (5), mm (5), mv (5).

NAME

mt – magnetic tape manipulating program

SYNOPSIS

mt [**–f** *tapename*] *command* [*count*]

DESCRIPTION

mt gives commands to a magnetic tape drive. If you do not specify a tape name, **mt** uses the environment variable TAPE; if TAPE does not exist, it uses the device **/dev/rmt12**. Note that *tapename* must reference a raw (not block) tape device. By default **mt** performs the requested operation once. Specify *count* if you want to perform operations more than once.

COMMANDS

The available commands are listed below. You need to specify only as many characters as are required to uniquely identify a command.

eof, weof	Write *count* end-of-file marks at the current position on the tape.
fsf	Forward space *count* files.
fsr	Forward space *count* records.
bsf	Backspace *count* files.
bsr	Backspace *count* records.
rewind	Rewinds the tape. Ignore *count*.
offline, rewoffl	Rewinds the tape and place the tape unit offline. Ignore *count*.
status	Prints status information about the tape unit.

DIAGNOSTICS

mt returns a 0 exit status when the operation(s) are successful; 1 if the command is unrecognized; and 2 if an operation fails.

FILES

/dev/rmt*	Raw magnetic tape interface

SEE ALSO

mtio(7), dd(1), ioctl(2), environ(5)

NAME

mv – move files

SYNOPSIS

mv [*–f*] *file1* [*file2* . . .] *target*
mv *dir1 dir2*

DESCRIPTION

Mv moves *file*(s) to a specified *target*. Under no circumstances can any of the *files* being manipulated be the same as the *target*, so take care when using shell metacharacters. If *target* is a directory, then the *file*(s) are moved to that directory. If *target* is a file, its old contents are replaced by the contents of *file*.

If **mv** determines that the mode of *target* forbids writing, it prints the mode, asks for a response, and reads the standard input for one line. If that line begins with y, the operation occurs if it is permissible; if not, **mv** exits. If the standard input is not a terminal, or if the **–f** (force) *option* is used, the **mv** is performed, if permitted, with no questions asked.

If *file1* is a directory, the directory rename occurs only if the two directories have the same parent; *file1 is renamed target.* If *file1* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file. If *target* is not a file, **mv** creates a new file with the same mode as *file1*. The owner and group of *target* are those of the user. If *target* is a file, moving a file into *target* does not change *target*'s mode, owner, or group. The **cp**(1) command sets the last modification time of *target*, (and last access time, if *target* did not exist). If *target* is a link to a file, all links remain and the file is changed.

OPTIONS

–f Forces the operation if it is permissable. Does not ask for confirmation.

SEE ALSO

chmod (1), cp (1), cpio (1), rm (1).

NAME

mmt, **mvt** – typeset documents, viewgraphs, and slides

SYNOPSIS

mmt [*options*] [*files*]

DESCRIPTION

Although very similar to **mm**(1), these two commands typeset their input via **troff**(1), as opposed to formatting it via **nroff**(1). The **mmt** command uses the MM macro package for its operations. The **mvt** command uses the Macro Package for View Graphs and Slides. Preprocessing through **tbl**(1) and **eqn**(1) is available for both. The proper pipelines and the required arguments and flags for **troff**(1) and the appropriate macro packages are generated, depending on the options selected.

Arguments or flags other than those given below are passed to **troff**(1) or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* argument. If no arguments are given, **mmt** prints a list of its options.

OPTIONS

–e	Invokes **eqn**(1) and cause it to read the */usr/pub/eqnchar* file. See **eqnchar**(5) for details concerning this file.
–t	Invokes **tbl**(1).
–T*dest*	Creates output for **troff**(1) device *dest*.
–Di10	Directs the output to the local Imagen Imprint-10 laser printer.
–a	Invokes the **–a** option of **troff**(1).
–y	Uses the noncompacted version of the macros. This is the default.
–z	Invokes no output filter to process or redirect the output of **troff**(1).

BUGS

If you specify simply a dash (–), –e option, and/or –t option along with the –o*list* option of **troff**(1), a harmless ''broken pipe'' diagnostic may result. This usually happens under these conditions if the last page of the document is not specified in *list*.

DIAGNOSTICS

m[mv]t: no input file

None of the arguments is a readable file and the command is not used as a filter.

SEE ALSO

env (1), eqn (1), mm (1), nroff (1), tbl (1), troff (1), environ (5), mm (5), mv (5).

NAME

netstat – show network status

SYNOPSIS

netstat [**–Aang**] [**–f** *address_family*]
netstat [**–himnrstT**] [**–f** *address_family*]
netstat [**–n**] [**–I** *interface*] *interval*

DESCRIPTION

The **netstat** command symbolically displays the contents of various network-related data structures. You can specify one of a number of output formats. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. The third form, with an *interval* specified, continuously displays the information regarding packet traffic on the configured network interfaces.

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form *host.port* or *network.port* if a socket's address specifies a network but no specific host address. It displays the host and network addresses, when known, symbolically, according to the databases /etc/**hosts** and /etc/**networks**, respectively. If a symbolic name for an address is unknown, or if you specify the **–n** option, **netstat** displays the address numerically, according to the address family. For more information regarding the Internet "dot format," refer to **inet**(3N). **netstat** displays unspecified, or "wildcard", addresses and ports an asterisk (*).

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. It also shows the network addresses of the interface and the maximum transmission unit (**mtu**).

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets.

The flags field shows the following:

- The state of the route (U if "up")
- Whether the route is to a gateway (G)
- Whether the route was created dynamically by a redirect (D)
- Whether the route has priority (P)
- Whether the route is a static (S) route added with **route**
- Whether the route has been marked for deletion (X).

Direct routes are created for each interface attached to the local host; the **gateway** field for such entries shows the address of the outgoing interface. The **refcnt** field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When you invoke **netstat** with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column for the primary interface (the first interface found during auto-configuration) and a column summarizing information for all interfaces. Use the –I option to replace the primary interface with another interface. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent output lines show values accumulated over the preceding interval.

OPTIONS

–A	With the default display, shows the address of any protocol-control blocks associated with sockets; used for debugging.
–a	With the default display, shows the state of all sockets; normally sockets used by server processes are not shown.
–g	With the default display, shows the first gateway used.
–h	Shows the state of the IMP host table. Status flags show the gateway entry (G), in use (U), a temporary entry (T).
–i	Shows the state of interfaces that were auto-configured (**netstat** does not show interfaces statically configured into a system, but not located at boot time).
–I *interface*	Shows information only about this interface; used with an *interval* as described below.
–m	Shows statistics recorded by the memory-management routines (the network manages a private pool of memory buffers).
–n	Shows network addresses as numbers (normally **netstat** interprets addresses and attempts to display them symbolically). You can use this option with any of the display formats.
–s	Shows per-protocol and routing statistics.
–r	Shows the routing tables.
–t	When used with the **-i** option, shows timer column.
–T	Shows all possible status information.
–f *address_family*	Limits statistics or address-control-block reports to those of the specified *address family*. The following address families are recognized: **inet**, for AF_INET; **ns**, for AF_NS; and **unix**, for AF_UNIX.

BUGS

The notion of errors is ill-defined. Collisions mean something else for the IMP.

SEE ALSO

hosts(4), networks(4), protocols(4), services(4), trpt(1M)

NAME

newform – change the format of a text file

SYNOPSIS

newform [–s] [–i *tabspec*] [–o *tabspec*] [–b *n*] [–e *n*] [–p *n*] [–a *n*] [–f] [–c *char*] [–l *n*] [*files*]

DESCRIPTION

newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect. Except for –s, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like "–e15 –l60" will yield results different from "–l60 –e15". Options are applied to all *files* on the command line.

OPTIONS

–s Shears off leading characters on each line up to the first tab and places up to eight of the sheared characters at the end of the line. If more than eight characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be: **newform** –s –i –l –a –e *filename*

–i*tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in **tabs**(1). In addition, if you specify a simple dash (–) for the value of *tabspec*, **newform** assumes that the tab specification is to be found in the first line read from the standard input (see **fspec**(4)). If no *tabspec* is given, *tabspec* defaults to –8. A *tabspec* of –0 expects no tabs; if any are found, they are treated as –1.

–o*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for –i*tabspec*. If no *tabspec* is given, *tabspec* defaults to –8. A *tabspec* of –0 means that no spaces will be converted to tabs on output.

–b*n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see –l*n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **–b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows: **newform –l1 –b7** *filename*

–e*n*	Same as **–b***n* except that characters are truncated from the end of the line.
–p*n*	Prefix *n* characters (see *–cchar*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.
–a*n*	Same as **–p***n* except characters are appended to the end of a line.
–f	Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **–o** option. If no **–o** option is specified, the line which is printed will contain the default specification of **–8**.
–c*char*	Change the prefix/append character to *char*. Default character for *char* is a space.
–l*n*	Set the effective line length to *n* characters. If *n* is not entered, **–l** defaults to 72. The default line length without the **–l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **–i** to expand tabs to spaces). The **–l1** option must be used to set the effective line length shorter than any existing line in the file so that the **–b** option is activated.

BUGS

The **newform** command normally only keeps track of physical characters; however, for the **–i** and **–o** options, **newform** will keep track of backspaces in order to line up tabs in the appropriate logical columns.

The **newform** command will not prompt the user if a *tabspec* is to be read from the standard input (by use of **–i––** or **–o––**).

If the **–f** option is used, and the last **–o** option specified was **–o––**, and was preceded by either a **–o––** or a **–i––**, the tab specification format line will be incorrect.

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*newform* was called with a bad option.
not –s format	There was no tab on one line.
can't open file	Self-explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.

tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

0 – normal execution
1 – for any error

SEE ALSO

csplit(1), tabs(1), fspec(4).

NAME

newgrp – log in to a new group

SYNOPSIS

newgrp [–] [*group*]

DESCRIPTION

The **newgrp** command changes your group identification. Although you remain logged in during the process, and your current directory is unchanged, **newgrp** sets new real and effective group IDs. The shell then performs calculations of access permissions to files with respect to these new IDs. You are always given a new shell to replace the current shell, regardless of whether **newgrp** terminates successfully or due to an error condition (e.g., unknown group).

Exported variables retain their values after you invoke **newgrp**. All unexported variables, however, are either reset to their default value or set to null. Unless you or the system itself exports system variables (e.g., PS1, PS2, PATH, MAIL, HOME), they are reset to default values. For example, suppose you have a primary prompt string (PS1) other than the default, a pound sign (#), and you have not exported PS1. After invoking **newgrp**, successfully or not, your PS1 variable is set to the default prompt string, the pound sign (#). Use the shell command, **export**, to export variables so that they retain their assigned value when invoking new shells. See **sh**(1) for more information.

With no arguments, **newgrp** changes the group identification back to the group specified in the your password file entry. If the first argument to **newgrp** is a dash (–), the environment changes to one that you would normally expect if you logged in again.

The **newgrp** command lets you change to any group of which you are a member. The /etc/**group** file contains a list of all groups and the group's members. You are a member of all groups for which you have an account. For example, if you have the following three registry accounts,

```
user1.project1.org
user1.project2.org
user1.project3.org
```

you are listed three times in the /etc/**group** file. You may not be listed in the group entry for your default group.

The /etc/**passwd** file contains your default group. Even though this may not appear in the /etc/**group** file, this group is always available as an option to the **newgrp** command.

BUGS

SysV does not allow group passwords.

FILES

/etc/group	System's group file
/etc/passwd	System's password file

SEE ALSO

login(1), sh(1), group(4), passwd(4), environ(5).

NAME

news – print news items

SYNOPSIS

news [**–a**] [**–n**] [**–s**] [*items*]

DESCRIPTION

The **news** program is used to keep the user informed of current events. By convention, these events are described by files in the directory, **/usr/news**.

When invoked without arguments, **news** prints the contents of all current files in /usr/news, most recent first, with each preceded by an appropriate header. The **news** program stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

Arguments other than the options listed below are assumed to be specific news items that are to be printed.

If **delete** is typed during the printing of a news item, printing stops and the next item is started. Another **delete** within one second of the first causes the program to terminate.

OPTIONS

–a Prints all items, regardless of currency. In this case, the stored time is not changed.

–n Reports the names of the current items without printing their contents, and without changing the stored time.

–s Reports how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of **news** in one's **.profile** file, or in the system's **/etc/profile**.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

profile(4), environ(5).

NAME

nice – run a command at low priority

SYNOPSIS

nice [– *increment*] *command* [*arguments*]

DESCRIPTION

The **nice** command executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., **−10**.

BUGS

An *increment* larger than 19 is equivalent to 19.

DIAGNOSTICS

The **nice** program returns the exit status of the subject command.

SEE ALSO

nohup(1), nice(2).

NAME

nl – line numbering filter

SYNOPSIS

nl [**–h** *type*] [**–b** *type*] [**–f** *type*] [**–v** *start#*] [**–i** *incr*] [**–p**] [**–l** *num*] [**–s** *sep*] [**–w** *width*] [**–n** *format*] [**–d** *delim*] *file*

DESCRIPTION

The **nl** command reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

The **nl** command views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body). The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	**Start of**
\:\:\:	Header
\:\:	Body
\:	Footer

Unless optioned otherwise, **nl** assumes the text being read is in a single logical page body. Command options may appear in any order and may be intermingled with an optional filename. Only one file may be named.

OPTIONS

–b*type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are:

–h*type* Same as –b*type* except for header. Default *type* for logical page header is **n** (no lines numbered).

a	Number all lines
t	Number lines with printable text only
p*string*	Number only lines that contain the regular expression specified in *string*.

n

Default *type* for logical page body is **t** (text lines numbered).

–f*type* Same as –b*type* except for footer. Default for logical page footer is **n** (no lines numbered).

−v*start#*	*Start#* is the initial value used to number logical page lines. Default is **1**.
−i*incr*	*Incr* is the increment value used to number logical page lines. Default is **1**.
−**p**	Do not restart numbering at logical page delimiters.
−l*num*	*Num* is the number of blank lines to be considered as one. For example, −l2 results in only the second adjacent blank being numbered (if the appropriate **−ha**, **−ba**, and/or **−fa** option is set). Default is **1**.
−s*sep*	*Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.
−w*width*	*Width* is the number of characters to be used for the line number. Default *width* is **6**.
−n*format*	*Format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes supressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).
−d*xx*	The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **−d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The following command will number **file1** starting at line number 10 with an increment of ten. The logical page delimiters are !+.

```
nl −v10 −i10 −d!+ file1
```

SEE ALSO

pr(1).

NAME

nm – print name list of common object file

SYNOPSIS

nm [*options*] *filename* . . .

DESCRIPTION

nm displays the symbol table of each common object file listed on the command line. The *filename* argument may refer to a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information is printed:

Name	The name of the symbol.
Value	Its value expressed as an offset or an address depending on its storage class.
Class	Its storage class.
Type	Its type and derived type. If the symbol is an instance of a structure or of a union then the structure or union tag will be given following the type (e.g., struct-tag). If the symbol is an array, then the array dimensions will be given following the type (e.g., char[n][m]). Note that the object file must have been compiled with the –g *option* of the cc(1) command for this information to appear.
Size	Its size in bytes, if available. Note that the object file must have been compiled with the –g *option* of the cc(1) command for this information to appear.
Line	The source line number at which it is defined, if available. Note that the object file must have been compiled with the –g *option* of the cc(1) command for this information to appear.
Section	For storage classes static and external, the object file section containing the symbol (e.g., text, data or bss).

The *options* listed below may be used in any order, either singly or in combination, and they may appear anywhere in the command line. Therefore, both **nm name** **–e** –v and **nm** –ve **name** print the static and external symbols in *name*, with external symbols sorted by value.

OPTIONS

–o	Prints the value and size of a symbol in octal instead of decimal.
–x	Prints the value and size of a symbol in hexadecimal instead of decimal.
–h	Does not display the output header data.
–v	Sorts external symbols by value before they are printed.
–n	Sorts external symbols by name before they are printed.
–e	Prints only external and static symbols.

−f Produces full output. Prints redundant symbols (.text, .data, .lib, and .bss), normally suppressed.

−u Prints undefined symbols only.

−r Prepends the name of the object file or archive to each output line.

−p Produces easily parsable, terse output. Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **S** (user defined segment symbol), **R** (register symbol), **F** (file symbol), **C** (common symbol), or **G** (global symbol) if −Ag is also specified. If the symbol is local (non-external), the type letter is in lowercase.

−V Prints the version of the **nm** command executing on the standard error output.

−T By default, **nm** prints the entire name of the symbols listed. Since object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The **−T** *option* causes *nm* to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

−Ag Checks in KGT (Known Global Table) to see if undefined globals are defined in global libraries. This affect the **−p** and **−u** *options*.

BUGS

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the −v and **−n** *options* should be used only in conjunction with the **−e** *option*.

FILES

TMPDIR/* Temporary files

TMPDIR is usually **/usr/tmp** but can be redefined by setting the environment variable TMPDIR [see **tempnam**() in **tmpnam**(3S)].

DIAGNOSTICS

nm: name: cannot open
name cannot be read.

nm: name: bad magic
name is not a common object file.

nm: name: no symbols
The symbols have been stripped from *name*.

SEE ALSO

cc(1), ld(1), tmpnam(3S), a.out(4), ar(4). %W% of %G% macro stdmacro

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

nohup *command* [*arguments*]

DESCRIPTION

The **nohup** command executes *command* with hangups and quits ignored. If output is not redirected by the user, both standard output and standard error are sent to a **nohup.out** file. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out** instead.

EXAMPLE

It is frequently desirable to apply **nohup** to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue this command:

```
nohup sh file
```

and the **nohup** applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type **sh** can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored (see **sh**(1)),asinthefollowing

```
nohup file &
```

Here is an example of what the contents of *file* could be:

```
sort ofile > nfile
```

WARNINGS

In the case of the following command, **nohup** applies only to *command1*:

```
nohup command1; command2
```

The command

```
nohup (command1; command2)
```

is syntactically incorrect.

SEE ALSO

chmod(1), nice(1), sh(1), signal(2).

NAME

nor.dan_to_iso – convert files to ISO format

SYNOPSIS

nor.dan_to_iso *input_file output_file*

DESCRIPTION

These utilities convert files written with the overloaded 7-bit national fonts to the International Standards Organization (ISO) 8-bit format. The overloaded fonts include any with a specific language suffix (for example, **f7x13.french**, or **din_f7x11.german**). If you created and/or edited a file using one of the national fonts, that file is a candidate for conversion.

You are not required to convert files, but probably will want to because

1. The overloaded fonts replace certain ASCII characters with national ones, have that subset of ASCII characters and the national characters in one file. The 8-bit fonts available as of SR10 include all the ASCII characters and the national characters.
2. The 8-bit fonts also include a wider range of national characters, so you can enter any character in any western European language. This was not always possible with the overloaded fonts. For example, there was not enough space in the overloaded font to include all the French characters, but they all exist in the 8-bit fonts.

There are two parameters to the conversion utilities. You must provide the name of the file you want to convert (*input_file*) and your *output_file*. If *output_file* already exists, the utilities abort.

The default location for the utilities is **/usr/apollo/bin**.

FILES

/usr/apollo/bin/french_to_iso	Converts overloaded French to ISO format
/usr/apollo/bin/german_to_iso	Converts overloaded German to ISO format
/usr/apollo/bin/nor.dan_to_iso	Converts overloaded Norwegian/Danish to ISO format
/usr/apollo/bin/swedish_to_iso	Converts overloaded Swedish/Finnish to ISO format
/usr/apollo/bin/swiss_to_iso	Converts overloaded Swiss to ISO format
/usr/apollo/bin/uk_to_iso	Converts overloaded U.K. English to ISO format

DIAGNOSTICS

All messages are generally self-explanatory.

NAME

obj2coff – convert OBJ format modules to COFF format modules

SYNOPSIS

obj2coff *objmodule coffmodule*

DESCRIPTION

The **obj2coff** command converts SR9.5 or later object format modules to SR10 COFF format modules. Either individual modules, or complete bound programs may be converted.

This command cannot be used to convert object module libraries, see **lbr2ar**(1) for that purpose.

BUGS

If **obj2coff** encounters an object module stamped with an SR8 systype (sys3, bsd4.1, or any SR8), it converts it to COFF but does not change the systype, and issues a warning:

```
module is stamped with obsolete systype 'systype_name'
```

Some UNIX system calls may behave differently between sys3 and sys5, or between bsd4.1 and bsd4.2, so users are cautioned to examine their programs carefully for any effects caused by changes in system call semantics.

For object format files, streams 2 and 3 are used for error input and error output, respectively. No error input stream is automatically assigned for COFF format files; stream 2 is assigned to error output. Thus an object file which has been converted to COFF format may not work if it attempts to read error input. Moreover, if it writes to error output, the error "operation attempted on unopened stream" will occur.

SEE ALSO

lbr2ar(1)

NAME

od – octal dump

SYNOPSIS

od [**–bcdosx**] [*file*] [[+] *offset* [**.**][**b**]]

DESCRIPTION

Od dumps *file* in one or more selected formats. If none of the options below are specified, *file* is interpreted in octal by default.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The *offset* argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If a period (.) is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by a plus sign (+).

Dumping continues until end-of-file.

OPTIONS

–b	Interpret bytes in octal.
–c	Interpret bytes in ASCII. Certain nongraphic characters appear as C escapes: null=\0, backspace=\b, formfeed=\f, newline=\n, return=\r, tab=\t; others appear as three-digit octal numbers.
–d	Interpret words in unsigned decimal.
–o	Interpret words in octal.
–s	Interpret 16-bit words in signed decimal.
–x	Interpret words in hexadecimal.

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [**–**] [**–f**] *name ...*

pcat *name ...*

unpack *name ...*

DESCRIPTION

Pack attempts to store the specified files in a compressed form. Wherever possible and useful, it replaces each input file *name* by a packed file *name.z* with the same access modes, access and modified dates, and owner as those of *name*.

If **pack** is successful, it removes *name*. Packed files can be restored to their original form using **unpack** or **pcat**.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each *.z* file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75 percent of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90 percent of the original size. **Pack** returns a value equaling the number of files not compressed.

No packing occurs if one or more of the following conditions exists:

File appears to be already packed.
Filename has more than 12 characters.
File has links.
File is a directory.
File cannot be opened.
No disk storage blocks will be saved by packing.
A file called *name.z* already exists.
.z file cannot be created.
An I/O error occurred during processing.

The last segment of the filename must contain no more than 12 characters to allow space for the appended *.z* extension. Directories cannot be compressed.

Pcat does for packed files what **cat**(1) does for ordinary files, except that **pcat** cannot be used as a filter. The specified files are unpacked and written to the standard output To view a packed file named *name.z* use:

pcat *name.z*

or just:

pcat *name*

To make an unpacked copy, say *nnn*, of a packed file named *name.z* (without destroying *name.z*) use the command:

> **pcat name > nnn**

Pcat returns the number of files it was unable to unpack, but will fail if one of the following conditions exist:

> Filename (exclusive of the *.z*) has more than 12 characters.
> File cannot be opened.
> File does not appear to be the output of **pack**.

Unpack expands files created by **pack**. For each file *name* specified in the command, a search is made for a file called *name.z* (or just *name*, if *name* ends in *.z*). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the *.z* suffix stripped from its name. It also has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. It will fail for the same reasons as those listed for **pcat**, as well as for the following additional reasons:

> a file with the ''unpacked'' name already exists
> the unpacked file cannot be created.

OPTIONS

(Note that these options are only for use with **pack**.)

–	Set an internal flag that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* cause the internal flag to be set and reset.
–f	Force packing of *name*. Useful for causing an entire directory to be packed even if some of the files will not benefit.

NOTES TO SysV USERS

The Apollo version of the **pack** command creates packed files that have an Apollo file type of ''uasc''. The original file type information is not carried over to the packed file. The **unpack** command checks the magic number of the unpacked file. If it matches one of the Apollo object types or archive type, the file type of the unpacked file is changed from ''uasc'' to ''obj''. If the file type of the original file is other than ''uasc'' or one of the ''obj'' types checked for by **unpack**, the file type must be manually changed after the file is unpacked.

SEE ALSO

cat (1).

NAME

chfn, **chsh**, **passwd** – change password file information

SYNOPSIS

passwd [–s] [**–f**] [*name*]
chsh *shell*
chfn

DESCRIPTION

The **passwd** command changes or installs a password, log-in shell (–s option), or GECOS information field (**–f** option) associated with the user *name* (your own name by default).

chsh changes a log-in shell, and is equivalent to **passwd –s**.

chfn changes the GECOS information field, and is equivalent to **passwd –f**.

When altering a password, **passwd** prompts for the current password and then for the new one; you must supply both. You must type the new password twice to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet, and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the super-user can change a password; owners must prove they know the old password.

When altering a log-in shell, (using **passwd** –s or **chsh**) the program displays the current log-in shell and then prompts for the new one. The new log-in shell must be one of the approved shells listed in **/etc/shells** unless you are the super-user. If **/etc/shells** does not exist, the only shells that can be specified are **/bin/sh**, **/bin/csh**, **/bin/ksh**, and **/com/sh**.

The super-user can change anyone's log-in shell; normal users can only change their own log-in shell(s).

When altering the GECOS information field, (using **passwd –f** or **chfn**), the program displays the current information, broken into fields, as interpreted by the **finger**(1) program (among others) and prompts for new values. These fields can include a user's "real life" name, office room number, office phone number, and home phone number. Each prompt includes a default value, which is enclosed between brackets. The default value is accepted simply by typing a carriage return. To enter a blank field, the word "none" can be typed. Phone numbers can be entered with or without hyphens. It is a good idea to run **finger** after changing the GECOS information to make sure everything is set up properly.

The super-user can change anyone's GECOS information; normal users can only change their own.

EXAMPLE

Below is a sample run:

```
% passwd -f
Name [Biff Studsworth II]:
Room number (Exs: 597E or 197C) []: 521E
Office Phone (Ex: 1632) []: 1863
Home Phone (Ex: 987532) [5771546]: none
```

NOTES

On Domain/OS systems, the **/etc/passwd** file is a typed file, which is automatically generated by the registry daemon. The registry administrator can make the person information in the registry read-only, in which case normal users cannot change the ''Name'' field.

FILES

/etc/passwd	The file containing all of this information
/etc/shells	The list of approved shells

SEE ALSO

login(1), finger(1), passwd(4), crypt(3C), edrgy(1M);
Using Your SysV Environment

NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste *file1 file2 ...*
paste –d *list file1 file2 ...*
paste –s [–d *list] file1 file2 ...*

DESCRIPTION

In the first two forms, **paste** concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of **cat**(1) which concatenates vertically, i.e., one file after the other. In the last form above, **paste** replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if a simple dash (-) is used in place of a filename.

OPTIONS

–d *list* Replace the default *tab* character by one or more alternate characters, specified in *list*. (see below). The *list* is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no –s option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (newline), **\t** (tab), \\ (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use *–d* "\\\\"

If *list* is not specified, the newline characters of each but the last file (or last line in case of the –s option) are replaced with a *tab* character.

–s Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **–d** option. Regardless of the *list*, the very last character of the file is forced to be a newline.

– May be used in place of any filename, to read a line from the standard input. (There is no prompting).

EXAMPLES

To list a directory in one column:

```
ls | paste –d" –
```

To list a directory in four columns:

```
ls | paste – – –
```

To combine pairs of lines into lines:

```
paste –s –d"\t\n" file
```

DIAGNOSTICS

line too long	Output lines are restricted to 511 characters.
too many files	Except in the case of the –s option, no more than 12 input files may be specified.

SEE ALSO

cut(1), grep(1), pr(1).

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [–f] *name* ...

pcat *name* ...

unpack *name* ...

DESCRIPTION

Pack attempts to store the specified files in a compressed form. Wherever possible and useful, it replaces each input file *name* by a packed file *name.z* with the same access modes, access and modified dates, and owner as those of *name*.

If **pack** is successful, it removes *name*. Packed files can be restored to their original form using **unpack** or **pcat**.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each *.z* file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75 percent of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90 percent of the original size. **Pack** returns a value equaling the number of files not compressed.

No packing occurs if one or more of the following conditions exists:

the file appears to be already packed
the filename has more than 12 characters
the file has links
the file is a directory
the file cannot be opened
no disk storage blocks will be saved by packing
a file called *name.z* already exists
the *.z* file cannot be created
an I/O error occurred during processing.

The last segment of the filename must contain no more than 12 characters to allow space for the appended *.z* extension. Directories cannot be compressed.

Pcat does for packed files what **cat**(1) does for ordinary files, except that **pcat** cannot be used as a filter. The specified files are unpacked and written to the standard output. To view a packed file named *name.z* use:

pcat *name.z*

or just:

pcat *name*

To make an unpacked copy, say *nnn*, of a packed file named *name.z* (without destroying *name.z*) use the command:

pcat name > nnn

Pcat returns the number of files it was unable to unpack, but will fail if one of the following conditions exist:

the filename (exclusive of the *.z*) has more than 12 characters
the file cannot be opened
the file does not appear to be the output of **pack**.

Unpack expands files created by **pack**. For each file *name* specified in the command, a search is made for a file called *name.z* (or just *name*, if *name* ends in *.z*). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the *.z* suffix stripped from its name. It also has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. It will fail for the same reasons as those listed for **pcat**, as well as for the following additional reasons:

a file with the ''unpacked'' name already exists
the unpacked file cannot be created.

OPTIONS

(Note that these options are only for use with **pack**.)

–	Sets an internal flag that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* cause the internal flag to be set and reset.
–f	Forces packing of *name*. Useful for causing an entire directory to be packed even if some of the files will not benefit.

NOTES TO SysV USERS

The Apollo version of the **pack** command creates packed files that have an Apollo file type of ''uasc''. The original file type information is not carried over to the packed file. The **unpack** command checks the magic number of the unpacked file. If it matches one of the Apollo object types or archive type, the file type of the unpacked file is changed from ''uasc'' to ''obj''. If the file type of the original file is other than ''uasc'' or one of the ''obj'' types checked for by **unpack**, the file type must be manually changed after the file is unpacked.

SEE ALSO

cat (1).

NAME

pg – file perusal filter for CRTs

SYNOPSIS

pg [*–number*] [**–p** *string*] [**–cefns**] [*+linenumber*] [*+/pattern/*] [*files* ...]

DESCRIPTION

The **pg** program is a filter which allows the examination of *files* one screenful at a time on a CRT. If you use a simple dash (–) and/or NULL arguments in place of a filename, **pg** reads from the standard input. Each screenful is followed by a prompt. If you hit a carriage return, **pg** displays another page; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, **pg** scans the **terminfo**(4) database for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The responses that may be typed when **pg** pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

Perusal Commands

(+1)<*newline*> or <*blank*>
: Displays one page. The address is specified in pages.

(+1) **l**
: With a relative address,simulates scrolling the screen, forward or backward, the number of lines specified. With an absolute address, prints a screenful beginning at the specified line.

(+1) **d** or **^D**
: Simulates scrolling half a screen forward or backward.

Perusal Commands That Take No Address

. or ^L
: Redisplays the current page of text.

$
: Displays the last windowful in the file. Use with caution when the input is a pipe.

Commands for Searching Text Patterns

The following commands are available for searching for text patterns in the text. The regular expressions described in **ed**(1) are available. They must always be terminated by a <*newline*>, even if the **–n** option is specified.

i/pattern/ Searches forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i^pattern^*
i?pattern?
Searches backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The circumflex (^) notation is useful for Adds 100 terminals which will not properly handle the question mark (?).

After searching, **pg** will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

Commands That Modify the Environment of

*i*n Begins perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i*p Begins perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i*w Displays another window of text. If *i* is present, set the window size to *i*.

s*filename* Saves the input in the named file. Only the current file being perused is saved. The white space between the s and *filename* is optional. This command must always be terminated by a <*newline*>, even if the **–n** option is specified.

h Helps by displaying an abbreviated summary of available commands.

q or Q Quits *pg*.

!*command* Passes *command* to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a <*newline*>, even if the **–n** option is specified.

At any time when output is being sent to the terminal, you can hit the quit key (normally CTRL-\) or the interrupt (break) key. This causes **pg** to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then **pg** acts just like **cat**(1), except that a header is printed before each file (if there is more than one).

OPTIONS

−*number*	Uses *number* to specify the size (in lines) of the window that **pg** is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).
−p *string*	Uses *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is a colon (:).
−c	Homes the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the **terminfo**(4) database.
−e	Refrains from pausing at the end of each file.
−f	Inhibits the splitting of lines. (Normally, **pg** splits lines longer than screen width, but some character sequences in the text being displayed (e.g., escape sequences for underlining) generate undesirable results.)
−n	Causes an automatic end of command as soon as a command letter is entered. (Normally, commands must be terminated by a *<newline>* character.)
−s	Prints all messages and prompts in standout mode (usually inverse video).
+*linenumber*	Starts up at *linenumber*.
+/*pattern*/	Starts up at the first line containing the regular expression *pattern.*

EXAMPLE

A sample usage of **pg** in reading system news would be

```
news | pg −p (Page %d):
```

NOTES

While waiting for terminal input, **pg** responds to **BREAK, DEL,** and ^ by terminating execution. Between prompts, however, these signals interrupt **pg**'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's **more**(1) will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using **pg** as a filter with another command that changes the terminal I/O options, terminal settings may not be restored correctly.

FILES

/usr/lib/terminfo/?/*	Terminal information database
/tmp/pg*	Temporary file when input is from a pipe

SEE ALSO

ed(1), grep(1). terminfo(4).

NAME

pr – print files

SYNOPSIS

pr [[–*column*] [–wwidth] [–a]] [–e*ck*] [–i*ck*] [**–drtfp**] [+*page*] [–n*ck*]
[–o*offset*] [–l*length*] [–s*separator*] [**–h** *header*] [*file* ...]

pr [[–m] [–w*width*]] [–e*ck*] [–i*ck*] [**–drtfp**] [+*page*] [–n*ck*]
[–o*offset*] [–l*length*] [–s*separator*] [**–h** *header*] *file1 file2* ...

DESCRIPTION

The **pr** command formats and prints the contents of a file. If you specify a simple dash (–) in place of *file*, or if you specify no files, **pr** assumes standard input. It prints the named files on standard output.

By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file. Page length is 66 lines which includes 10 lines of header and trailer output. The header is composed of 2 blank lines, 1 line of text (can be altered with –**h**), and 2 blank lines; the trailer is 5 blank lines. For single column output, line width may not be set and is unlimited. For multicolumn output, line width may be set and the default is 72 columns. Diagnostic reports (failed options) are reported at the end of standard output associated with a terminal, rather than interspersed in the output. Pages are separated by series of line feeds rather than form feed characters.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the –s option is used, lines are not truncated and columns are separated by the *separator* character.

Either –*column* or –**m** should be used to produce multi-column output. The –**a** option should only be used with –*column* and not –**m**.

OPTIONS

+*page*	Begin printing with page numbered *page* (default is 1).
–*column*	Print *column* columns of output (default is 1). Output appears as if –**e** and –**i** are turned on for multi-column output. May not use with **-m**.
–**a**	Print multi-column output across the page one line per column. The value of *columns* must be greater than one. If a line is too long to fit in a column, it is truncated.
–**m**	Merge and print all files simultaneously, one per column. The maximum number of files that may be specifed is eight. If a line is too long to fit in a column, it is truncated. May not use with –*column*.

−d	Double-space the output. Blank lines that result from double-spacing are dropped when they occur at the top of a page.
−e*ck*	Expand input tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).
−i*ck*	In output, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).
−n*ck*	Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of single column output or each line of **−m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).
−w*width*	Set the width of a line to *width* character positions (default is 72). This is effective only for multi-column output (*-There is no line limit for single column output.*
−o*offset*	Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the width and offset.
−l*length*	Set the length of a page to *length* lines (default is 66). The −l0 argument is reset to −l66. When the value of *length* is 10 or less, **−t** appears to be in effect since headers and trailers are suppressed. By default, output contains 5 lines of header and 5 lines of trailer leaving 56 lines for user-supplied text. When *−llength* is used and *length* exceeds 10, then *length*-10 lines are left per page for user supplied text. When *length* is 10 or less, header and trailer output is omitted to make room for user supplied text.

−h *header* Use *header* as the text line of the header to be printed instead of the file name. **−h** is ignored when **−t** is specified or −l*length* is specified and the value of *length* is 10 or less. (**−h** is the only **pr** option requiring space between the option and argument.)

−p Pause before beginning each page if the output is directed to a terminal (**pr** will ring the bell at the terminal and wait for a carriage return).

−f Use single form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

−r Print no diagnostic reports on files that will not open.

−t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page. Use of **−t** overrides the **−h** option.

−s*separator* Separate columns by the single character *separator* instead of by the appropriate number of spaces (default for *separator* is a tab). Prevents truncation of lines on multicolumn output unless -w is specified.

EXAMPLES

To print file1 and file2 as a double-spaced, three-column listing headed by "file list", type the following:

```
pr −3dh "file list" file1 file2
```

To copy file1 to file2, expanding tabs to columns 10, 19, 28, 37, ... use this command line:

```
pr −e9 −t <file1 >file2
```

To print file1 and file2 simultaneously in a two-column listing with no header or trailer where both columns have line numbers, type this:

```
pr −t -n file1 | pr
```

FILES

/dev/tty*
to delay messages enabling them to print at the bottom of files rather than interspersed throughout printed output.

SEE ALSO

cat(1), pg(1).

NAME

prf – queue a file for printing by Domain/OS Aegis print spooler

SYNOPSIS

prf [*options*] *pathname...*

DESCRIPTION

The **prf** command queues a file for printing. The file must be an ASCII stream (that is, text) file, a graphics map file (GMF), or a GPR bitmap object. After successfully queuing a file, **prf** displays a message containing the full pathname of the file that you queued.

You can execute **prf** once for each file that you want to print (specifying all the necessary options every time), or you can enter **prf**'s interactive mode and hand files to the program continuously. See the examples for a sample interactive session.

Files queued by **prf** are physically printed by **prsvr**, the print server, running as a background task under control of **prmgr**, the print manager.

When you invoke **prf**, it first sets all options to their default states. Next, it looks for the print options file called **user_data/startup.prf** unless you invoke **prf** with the **–ndb** option. If **prf** locates the option file, it executes the options contained in the file to configure the current session. Finally, it processes all options on the command line.

pathname (optional)	Specify the file to be printed. Multiple pathnames and pathname wildcarding are permitted.
	Default if omitted: read standard input

OPTIONS

The following options can appear on the shell command line or in **prf** interactive mode. In addition, you can place one or more options in a **prf** option file so that they are executed automatically whenever you invoke **prf**.

Many of the options have default values that are specified in the **prsvr** configuration file established for each printer in the network by the system administrator. If you omit these options, your file is printed using the values specified in the **prsvr** configuration file. For example, omission of the **–banner** option could cause your file to be printed with a banner page if the **prsvr** configuration file specifies one.

If no options are specified, the file is printed using ASCII carriage control, with pagination enabled, on the default printer as established by the system administrator.

Options Applying to All File Types

–inter[active] Enter interactive mode.

–sea[rch_dir] {on|off}
Searches through all the directories of all the active processes on your node for the file(s) to be printed. This option is most useful in interactive mode, when the working directory of the **prf** process may be different from the working directory of the file to be printed.

The default is **off**.

–cop[ies] *n* Prints multiple copies of the file, where *n* is the requested number of copies. If **–cop[ies]** is specified, *n* is required. The default is one copy.

–pr[inter]*name* Specifies the name of the printer that should print the file. This option is useful only if more than one printer is in use on the network, or if a printer has been assigned a nonstandard name with the **printer_name** configuration directive in the **prsvr** command. If you omit this option, **prf** uses the default printer name, **p**. Note that **p** is also the default printer name used by the print server.

–s[ite] *spool_node_name*
Uses this option only if you are queuing jobs to a pre-SR10 print server connected to a spool directory (/sys/**print**) that is different from the one specified by your node. By default, SR10 printers find the spool node for you.

–nc[opy] Prints the specified file from its location in the user-specified directory, bypassing /sys/**print**/**spooler**. If you select this option, **prf** defaults to the no-delete (**–nd**) option. If you specify the delete (**–d**) option, the file is deleted at the completion of the print request. If you use this option (with or without the delete option), do not open and alter the print file before the print job is completed.

–d[elete] (default) Deletes the print file at the completion of the print job.

–nd[elete] Does not delete the print file when the print server is finished printing it. This becomes default if **–nc** is specified.

–user[*username*] Specifies the user name that appears on the banner page of the printed file. The alarm facility of **prf** also uses this name to determine who should be notified when printing is complete (see **–sig** below). This means that this name must be a valid log-in name (unless you don't care about sending an alarm).

The default is the current log-in name.

–sig[nal] {alarm|off} Requests an alarm server signal when the file has finished printing.

The default is **off**.

–ban[ner] [on|off] Enables/disables banner page. If the banner setting in the **prsvr** configuration file is **off**, no banner is printed.

The default is **on**.

–config[_file] [*pathname*]
Specifies a file containing further **prf** options, one per line. Do not use prefixed hyphens (-) with the option names in the configuration file. If *pathname* is omitted, **prf** executes the **prf** option file **/user_data/startup_prf**.

–ndb Suppresses processing of the **prf** option file.

–trans[parent] [on|off]
off specifies that the file being printed is passed directly to the printer driver routine with no processing by the print server. The default is **on**.

–filter[_chain] *string*
Specifies a filter string that will be used by the print server to process the job. This option overrides the default processing done by the print server. It is most often used to invoke filters that have been added to the print server. The format of the string is "filter1 | filter2", where filter1 and filter2 are composed of strings of the form "type1$type2" and "type2$type3". Note that the output type of filter *n* must equal the input type of filter *n*+1 .

–paper_size {a|b|legal|a3|a4|a5|b4|b5}
Selects the paper size. You must specify one of the following size codes:

Code	Size in inches (mm)
a	8.50 x 11.00
b	11.00 x 17.00
legal	8.50 x 14.00
a3	11.69 x 16.54 (297mm x 420mm)
a4	8.27 x 11.69 (210mm x 297mm)
a5	5.38 x 8.27 (137mm x 210mm)
b4	9.84 x 13.90 (257mm x 364mm)
b5	5.93 x 9.89 (182mm x 257mm)

TILDE ESCAPES.if 0=0 .nr c. 38636-0-13

This option is available only for the Domain/Laser-26 and APPLE LaserWriter* printers. Because **prf** assumes that the correct paper is in the printer's paper tray, you should check the paper tray before printing. The default paper size is specified in the **prsvr** configuration file.

Options Applying to Text Files Only

–margins [on|off] Enables/disables margins generated by **prf**.

The default is **on**.

–top *n* Specifies top page margin, in inches. The default is a value specified in the **prsvr** configuration file.

–bot[tom] *n* Specifies bottom page margin, in inches. The default is a value specified in the **prsvr** configuration file.

–right *n* Specifies right margin, in inches. The default is 0 inches.

–left *n* Specifies left margin, in inches. The default is 0 inches.

–headers [on|off] Enables/disables page headers and footers generated by **prf**. The default is **on**.

–head[_string] *l-string/c-string/r-string*

Specifies contents of left, center, and right components of the page header generated by **prf**. Components can be empty strings. The following special characters return the values indicated when they appear in the header strings:

Character	**Return Value**
@	= Escape character
#	= current Page number with 1 leading and 1 trailing space
%	= Current date
!	= Filename
&	= Filename's last time, date modified
*	= Insert a space in text string (literal spaces are not allowed)

TILDE ESCAPES.if 0=0 .nr c. 38842-0-15

Example: **–head** !/Page#/% produces a header with the filename in the left component, the string "Page" followed by the current page number in the center component, and the current date in the right component. The default header is a string specified in the **prsvr** configuration file.

–foot[_string] *l-string/c-string/r-string*
Specifies contents of page footers. The format is the same as for **–head** above. There is no default footer.

–ftn [on|off] Enables/disables FORTRAN carriage control. **–ftn on** causes the print server to use FORTRAN forms control even if the file does not have the FORTRAN carriage-control flag. Use of this option causes **prf** to interpret the first character of each line as a FORTRAN carriage control character (and not print it). This can be unfortunate if the file has ASCII carriage control, so be careful. **–ftn off** causes the print server to print the contents of column one rather than trying to interpret it as FORTRAN forms control. If this option is specified without **on** or **off**, **on** is assumed.

The default is **off**.

–wrap [on|off] Enables/disables automatic line wrapping. When enabled, **prf** wraps lines that exceed the right margin. When disabled, **prf** truncates lines that exceed the right margin. If this option is specified without **on** or **off**, **on** is assumed.

The default is **off**.

–col[umns] {1|2} Specifies single-or double-column printing.

The default state is single column.

–lpi *n* Specifies the line-spacing factor. *n* is an integer indicating the number of lines per inch.

The default is six lines per inch.

Options for Variable Font and Pitch

–pitch *n* Sets the printer pitch (characters/inch). The following pitch settings are available on the printers indicated.

Printer	**Pitch**
Printronix*	10
Spinwriter*	12
IMAGEN*	8.5, 10, 12, 15, 17.1
GENICOM*	10, 12, 13.1, 16.7
Versatec*	12
LaserWriter*	1 to 100
Laser-26	1 to 100

TILDE ESCAPES.if 0=0 .nr c. 39004-0-12

–point *n*
: Sets the point size for the font to be used. This is a real number that specifies size in points. A point equals 1/72 inch.

–weight {light|medium|bold}
: Sets the weight of the font to be used.

  The default is **medium**.

–lq [on|off]
: Specifies that the document is to be printed in letter quality (**on**) or in draft (**off**) mode. With no argument, **on** is assumed when this option is invoked. If the option is not invoked, draft mode is the default.

Options Applying to Plot Files

–res[olution] *n*
: Specifies output plot resolution in dots per inch. If you specify a resolution not available on the particular printer, **prsvr** prints the file at the closest available resolution.

  The default resolution is specified in the **prsvr** configuration file.

–white[_space] *n*
: Specifies the amount of white space (in inches) to appear between multiple plots in one file.

  The default is three inches.

–bw[_rev] [on|off]
: Enables/disables black and white reversal for bitmaps. If no argument is specified, **on** is assumed. If the option is not invoked, black/white reversal is disabled.

-magn[ification] *n*
: Specifies bitmap magnification value. *n* is an integer in the range -1 to 16. The values have the following meanings:

  -1 Selects auto-scaling to magnify the bitmap to fill the available page space.

  0 Selects one-to-one scaling between the display and the printer for GMF bitmaps. (For GPR bitmaps, this translates to magnification 1.)

  1-16 Selects the magnification indicated by value. Where 1 equals 1-to-1, 2 equals 2x, etc.

  Default if omitted: *n* is 0

Options Applying to PostScript* Printers

The following options apply only for files sent to printers that contain the PostScript interpreter, such as the Domain/Laser-26 and APPLE LaserWriter* printers.

–post[script] [on|off]

Enables/disables PostScript interpretation. When enabled, the data is passed through the PostScript interpreter. When disabled, the data is printed as text, plot, or transparent data. If the option is not invoked, PostScript interpretation is disabled.

The default is **on**.

–orient[ation] {port[rait]|land[scape]}

Selects the page orientation. **portrait** specifies that the text or x-axis of the bitmap is printed parallel to the short edge of the paper. **landscape** specifies that the text or x-axis of the bitmap is printed parallel to the long edge of the paper and perpendicular to the short leading edge.

The default is **portrait**.

Information Request Options

–check [**–pr** *printer_name*]

Checks for the existence of the specified printer. If the printer does not exist or is unavailable, an error message is returned.

–list_printers Lists the names and status of all printers currently attached to the network.

–list_sites Lists the names of all print managers currently in the network.

–sig_printer *printer_name* {**-abort**|**-sus[pend]**|**cont[inue]**}

Signals the printer to abort, suspend, or continue an active print job.

–pre10 Allows you to queue print requests to a pre-SR10 print server.

COMMANDS

Once **prf** has been invoked in interactive mode (see **–inter** above), it accepts the following interactive commands at the "prf> " prompt (in addition to the options already discussed).

p[rint] [*print_file_pathname*] [*options*]

Queue the specified file for printing.

q[uit] Quit interactive mode and return to the shell.

sh[ell]	Create a shell command line. This command allows you to issue shell commands without leaving **prf** interactive mode. When you finish entering shell commands, type CTRL/Z. This returns you to **prf** interactive mode. Your previous **prf** option settings remain undisturbed by the intervening shell commands.
init[ialize]	Reset **prf** parameters to their default values.
r[ead] [*printer*]	List queue entries for the specified printer. If *printer* is omitted, the contents of the queue (determined by the current setting of **−pr**) are listed.
wd [*pathname*]	Execute the shell command **wd** (working_directory) to set or display the working directory.
get *option*	Display the value of the **prf** option specified. Use this command to show the settings of the various **prf** parameters.
can[cel] [*job_id*]	Cancel printing of the specified file at the current printer. Note that you must specify the job ID assigned by **prmgr** when the file is queued. Use the **read** command to display the names and job IDs of currently queued files. This command affects jobs in the print queue; it does not cancel a job being printed. To halt a job being printed, use **−pr_sig** with abort specified.

EXAMPLES

The following example, queues the file named **mary** for printing and forces FORTRAN carriage returns:

```
$ prf mary -ftn
"//nodel/my_dir/mary" queued for printing.
$
```

The following example queues the file named **filex** to the printer queue on the node named **//tape**:

```
$ prf filex -s //tape
  "//nodel/my_dir/test_file.pas" queued for printing at site //tap
$
```

This example shows the types of commands that might appear in the default **prf** configuration file **/user_data/startup.prf**:

```
pr ge
site //rye
foot %/my_file/&
```

The following example shows a sample interactive session:

```
$ prf -inter
prf> get pr
pr = p
prf> -pr cx
prf> get pr
pr = cx
prf> -pitch 20
prf> print test_file.pas
"//node1/my_dir/test_file.pas" queued for printing.
prf> q
$
```

This example illustrated running **prf** from an icon. To run **prf** interactively in a process devoted to it, insert the following command in the start-up file that you use to start the DM:

cp -i -c 'P' /com/prf -inter -n print_file

The above command creates a **prf** process and turns its window into an icon using the print icon character in (**/sys/dm/fonts/icons**). Issue the DM command **icon** to change the icon window into its full-size format.

NOTES

APPLE and LaserWriter are registered trademarks of Apple Computer, Inc.
Printronix is a trademark of Printronix, Inc.
Spinwriter is a registered trademark of NEC, Inc.
IMAGEN is a registered trademark of IMAGEN Corp.
GENICOM is a registered trademark of GENDICOM Corp.
Versatec is a trademark of Versatec, Inc.
PostScript is a registered trademark of Adobe Systems, Inc.

NAME

prof – display profile data

SYNOPSIS

prof [–tcan] [–ox] [–g] [–z] [–h] [–s] [–m *mdata*]" [*prog*]

DESCRIPTION

The **prof** command interprets a profile file produced by the **monitor**(3C) function. The symbol table in the object file *prog* (**a.out** by default) is read and correlated with a profile file (**mon.out** by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options **t, c, a,** and **n** determine the type of sorting of the output lines. The mutually exclusive options **o** and **x** specify the printing of the address of each symbol monitored. All other options may be used in any combination.

A program creates a profile file if it has been loaded with the **–p** option of **cc**(1). This option to the **cc** command arranges for calls to **monitor**(3C) at the beginning and end of execution. It is the call to **monitor** at the end of execution that causes a profile file to be written. The number of calls to a function is tallied if the **–p** option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable PROFDIR. If PROFDIR does not exist, ''mon.out'' is produced in the directory that is current when the program terminates. If PROFDIR = string, ''string/pid.progname'' is produced, where *progname* consists of argv[0] with any path prefix removed, and *pid* is the program's process id. If PROFDIR is the null string, no profiling output is produced.

A single function may be split into subfunctions for profiling by means of the MARK macro [see **prof**(5)].

OPTIONS

–t	Sort by decreasing percentage of total time (default).
–c	Sort by decreasing number of calls.
–a	Sort by increasing symbol address.
–n	Sort lexically by symbol name.
–o	Print each symbol address (in octal) along with the symbol name.
–x	Print each symbol address (in hexadecimal) along with the symbol name.
–g	Include non-global symbols (static functions).
–z	Include all symbols in the profile range [see **monitor**(3C)], even if associated with zero number of calls and zero time.

−h	Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)
−s	Print a summary of several of the monitoring parameters and statistics on the standard error output.
−m *mdata*	Use file *mdata* instead of **mon.out** as the input profile file.

FILES

mon.out	For profile
a.out	For namelist

WARNINGS

The times reported in successive identical runs may show variances of 20% or more, because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may "beat" with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely.

The times for static functions are attributed to the preceding external text symbol if the −g option is not used. However, the call counts for the preceding function are still correct, i.e., the static function call counts are not added in with the call counts of the external function.

CAVEATS

Only programs that call **exit**(2) or return from *main* will cause a profile file to be produced, unless a final call to monitor is explicitly coded.

The use of the **−p** option to cc(1) to invoke profiling imposes a limit of 600 functions that may have call counters established during program execution. For more counters you must call **monitor**(3C) directly. If this limit is exceeded, other data will be overwritten and the **mon.out** file will be corrupted. The number of call counters used will be reported automatically by the **prof** command whenever the number exceeds 5/6 of the maximum.

SEE ALSO

cc(1), exit(2), profil(2), monitor(3C), prof(5).

NAME

prs – print an SCCS file

SYNOPSIS

prs [**–d**[*dataspec*]] [**–r**[*SID*]] [**–e**] [**–l**] [**–c**[*date-time*]] [**–a**] *file* ...

DESCRIPTION

prs prints, on the standard output, part or all of an SCCS (Source Code Control System) file in a user-supplied format. If you name a directory, **prs** behaves as though each file in the directory is specified as a named file, except that it silently ignores non-SCCS and unreadable files. If a dash (–) is given in place of a filename, **prs** reads the standard input, taking each line to be the name of an SCCS file or directory to be processed. Options to **prs** may appear in any order. Each argument applies independently to each named file.

OPTIONS

–d[*dataspec*]	Specify the output data specification. The *dataspec* is a string consisting of SCCS file data keywords interspersed with optional user-supplied text.
–r[*SID*]	Specify the SCCS Identification (SID) string of the delta for which information is desired. If you do not specify an SID, then **prs** assumes the SID to be that of the most recently created delta.
–e	Request information for all deltas created earlier than and including the delta designated via the **–r** keyletter or the date given by the **–c** option.
–l	Request information for all deltas created later than and including the delta designated via the **–r** keyletter or the date given by the **–c** option.
–c[*date-time*]	Specify *date-time* as cutoff for requesting information. This cutoff *date-time* appears in the following form: YY[MM[DD[HH[MM[SS]]]]] Units omitted from the *date-time* default to their maximum possible values; that is, **–c8502** is equivalent to **–c850228235959**. Any number of non-numeric characters may separate the various two-digit pieces of the cutoff date in the following form: **–c85/2/2 9:22:25**.
–a	Request printing of information for both removed (delta type = **R**) and existing (delta type = **D**) deltas. Refer to **rmdel**(1) for more information. If you do not specify the **–a** keyletter, **prs** provides information only on existing deltas.

DATA KEYWORDS

Data keywords specify those parts of an SCCS file to be retrieved and output. All parts of an SCCS file have an associated data keyword. Refer to sccsfile(4) for more information about the structure of these file types. There is no limit on the number of times a data keyword may appear in a *dataspec*.

prs prints the user-supplied text, and appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either "Simple" (S), in which keyword substitution is direct, or "Multiline" (M), in which keyword substitution is followed by a carriage return. User-supplied text is any text other than recognized data keywords.

A tab is specified by \t, and a carriage return/newline is specified by \n. The default data keywords are: "**:Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:**"

SCCS Files Data Keywords

Keyword	Data Item	File Section	Value	Format
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	D or R	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:m:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S

SCCS Files Data Keywords (Contd.)

Keyword	Data Item	File Section	Value	Format
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	yes or no	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	yes or no	S
:BF:	Branch flag	"	yes or no	S
:J:	Joint edit flag	"	yes or no	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	yes or no	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of **what**(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of **what**(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	**what**(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

% prs –d"Users and/or user IDs for :F: are:0UN:" s.file

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

% prs –d"Newest delta for pgm :M:: :I: Created :D: By :P:" –r s.file

```
Newest delta for pgm main.c: 3.7 Created 85/12/1 By cas
```

A simple command line without options, such as **prs s.file**, may produce the following on the standard output, for each delta table entry of the "D" type:

```
D 1.1 85/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
b178-12345
b179-54321
COMMENTS:
this is the comment line for s.file initial delta
%
```

FILES

/tmp/pr?????

DIAGNOSTICS

Use **help**(1) for explanations.

SEE ALSO

admin(1), delta(1), get(1), help(1), sccs(1), sccsfile(4);
Using Your SysV Environment.

NAME

ps – report process status

SYNOPSIS

ps [*options*]

DESCRIPTION

The **ps** command prints information about active processes. Without *options*, information is printed about processes associated with the controlling terminal. The output is a list consisting of the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

All *options* accept names or lists as arguments. Arguments can be either separated from one another by commas or enclosed in double quotes and separated from one another by commas or spaces. Values for *proclist* and *grplist* must be numeric.

Under the **–f** option, **ps** tries to determine the command name and arguments given when the process was created by examining the user block. Failing this, the command name is printed, as it would have appeared without the **–f** option, in square brackets.

The column headings and the meaning of the columns in a **ps** list are given below; the letters **f** and **l** indicate the option (full or long, respectively) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do not determine which processes will be listed.

F	(l)	Flags (hexadecimal and additive) associated with the process
S	(l)	The state of the process:
UID	(f,l)	The user ID number of the process owner (the login name is printed under the **–f** option).
PID	(all)	The process ID of the process (this datum is necessary in order to kill a process).
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilization for scheduling.
PRI	(l)	The priority of the process (higher numbers mean lower priority).
NI	(l)	Nice value, used in priority computation.
ADDR	(l)	The memory address of the process.
SZ	(l)	The size (in pages or clicks) of the swappable process's image in main memory.
WCHAN	(l)	The address of an event for which the process is sleeping, or in SXBRK state, (if blank, the process is running).

STIME (f)	The starting time of the process, given in hours, minutes, and seconds. (A process begun more than twenty-four hours before the *ps* inquiry is executed is given in months and days.)
TTY (all)	The controlling terminal for the process (the message, ?, is printed when there is no controlling terminal).
TIME (all)	The cumulative execution time for the process.
COMMAND(all)	The command name (the full command name and its arguments are printed under the **–f** option).

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

OPTIONS

The *options* are given in descending order according to volume and range of information provided:

–e	Print information about every process now running.
–d	Print information about all processes except process group leaders.
–a	Print information about all processes most frequently requested: all those except process group leaders and processes not associated with a terminal.
–f	Generate a full list. (See below for significance of columns in a full list.)
–l	Generate a long list. (See below.)
–t *termlist*	List only process data associated with the terminal given in *termlist*. Terminal identifiers may be specified in one of two forms: the device's file name (e.g., **tty04**) or, if the device's file name starts with **tty**, just the digit identifier (e.g., **04**).
–p *proclist*	List only process data whose process ID numbers are given in *proclist*.
–u *uidlist*	List only process data whose user ID number or login name is given in *uidlist*. In the listing, the numerical user ID will be printed unless you give the **–f** option, which prints the login name.
–g *grplist*	List only process data whose process group leader's ID number(s) appears in *grplist*. (A group leader is a process whose process ID number is identical to its process group ID number. A login shell is a common example of a process group leader.)

NOTE

You can perform a **ps** on a remote node. To do this, use the following syntax:

/bin/ps [*options*] **–n** [*nodename*]

FILES

/dev	
/dev/tty*	
/etc/passwd	UID information supplier

WARNING

Things can change while **ps** is running; the snap-shot it gives is only true for a split-second, and it may not be accurate by the time you see it. Some data printed for defunct processes is irrelevant.

If no *termlist*, *proclist*, *uidlist*, or *grplist* is specified, **ps** checks *stdin*, *stdout*, and *stderr* in that order, looking for the controlling terminal and attempts to report on processes associated with the controlling terminal. In this situation, if *stdin*, *stdout*, and *stderr* are all redirected, **ps** does not find a controlling terminal, and there is no report.

On a heavily loaded system, **ps** can report an **lseek**(2) error and exit. The **ps** program can seek to an invalid user area address; having gotten the address of a process' user area, **ps** cannot seek to that address before the process exits and the address becomes invalid.

Specifying **ps –ef** may not result in the reporting of the actual start of a tty login session; instead, an earlier time, when a getty was last respawned on the tty line, may be reported.

SEE ALSO

kill(1), nice(1), getty(1M).

NAME

ptx – permuted index

SYNOPSIS

ptx [*options*] [*input* [*output*]]

DESCRIPTION

The **ptx** command generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). First, it does the permutation, generating one line for each keyword in an input line; then it rotates the keyword to the front and sorts the permuted file; and finally, it rotates the sorted lines so the keyword comes at the middle of each line.

The output from **ptx** appears in the following form:

.xx "tail" "before keyword" "keyword and after" "head"

The *.xx* shown above is assumed to be an **nroff**(1) or **troff**(1) macro that you provide. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

OPTIONS

–f	Fold upper- and lowercase letters for sorting.
–t	Prepare the output for the phototypesetter.
–w *n*	Use *n* as the length of the output line. The default line length is 72 characters for **nroff**(1) and 100 for **troff**(1).
–g *n*	Use *n* as the number of characters to reserve for each gap among the four parts of the line as finally printed. The default gap is three characters.
–o *only*	Use as keywords only the words given in the *only* file.
–i *ignore*	Do not use as keywords any words given in the *ignore* file. If the **–i** and **–o** options are missing, use */usr/lib/eign* as the *ignore* file.
–b *break*	Use the characters in the *break* file to separate words. Tab, newline, and space characters are *always* used as break characters.
–r	Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter), separate from the text of the line. Attach that identifier as a fifth field on each output line.

BUGS

Line length counts do not account for overstriking or proportional spacing.

Because **ptx** uses tildes internally, lines containing them do not print correctly.

FILES

/bin/sort
/usr/lib/eign

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

The **pwd** command prints the pathname of the working (current) directory.

DIAGNOSTICS

Cannot open .. and *Read error in* ..

Indicate possible file system trouble and should be referred to a UNIX system administrator.

SEE ALSO

cd(1).

NAME

ratfor – rational FORTRAN dialect

SYNOPSIS

ratfor [*options*] [*files*]

DESCRIPTION

The **ratfor** command converts a rational dialect of FORTRAN into ordinary FORTRAN. It provides control flow constructs essentially identical to those in C, as well as simplified syntax to make programs easier to read and write. These constructs are described below:

```
        statement grouping:
                { statement; statement; statement }

        decision-making:
                if (condition) statement [ else statement ]
                switch (integer value) {
                        case integer:     statement
                        ...
                        [ default: ]      statement
                }

        loops:
                while (condition) statement
                for (expression; condition; expression) statement
                do limits statement
                repeat statement [ until (condition) ]
                break
                next
```

free form input:
    multiple statements/line; automatic continuation

comments:
    # this is a comment.

translation of relationals:
    >, >=, etc., become **.GT.**, **.GE.**, etc.

return expression to caller from function:
    return (*expression*)

define:
    define *name replacement*

include:
    include *file*

OPTIONS

−h	Turn quoted strings into 27H constructs.
−C	Copy comments to the output and attempt to format it neatly.
−6x	Make the continuation character *x* and place it in column six. Normally, continuation lines are marked with an ampersand (&) in column one.

NAME

rbak – restore or index a magnetic media backup file

SYNOPSIS

rbak {**–f** *fileno*|**–fid** *id*} [**–dev** | *m*[*unit*] | **f** | **ct**]
[**–int**|**–index**] [**–sla**|**–nsla**] [**–anys**] [**–reo**] [**–pr** *pn*]
[**–cr**|**–r**|**–ms**|**–md**] [**–force**] [**–du**] [**–l**|**–ld**|**–lf**|**–ll**]
[**–reten**|**–nreten**] [**–rewind**] [**–dacl**|**–sacl**]
[**–from** *filename*] [**–pdt**] [**–stdin**] {{**–all**|**pn**}
[**–as** *disk_pn*]}...

DESCRIPTION

rbak restores objects from the backup input media written by **wbak** (write_backup). The backup input media can be one of magnetic media, file or standard input.

Use **wbak** and **rbak** to back up disks and to transfer information between separate Domain installations. (Use the rwmt (read_write_magtape) command to transfer information to and from non–Domain installations.)

rbak operates in either index or interchange mode. To restore objects to disk, use interchange mode (**–int**). To list object names on standard output, without restoring any information to disk, use index mode (**–index**).

pathname (optional)	Specify name of the object to be indexed or restored to disk. This may be a directory, file, or link. If the object is being restored, the new disk object has the same name. If you wish the disk file to be saved under a different name, use –as (below). Multiple pathnames are permitted; however, wildcarding is not supported. Default if omitted: must use **–all**

OPTIONS

Backup File Identifiers

One of the following options is required.

–f *file_no*	Reads the back up file with the file number specified. You assigned this number with **wbak**.
–f *cur*	Begins reading at current position on the back up medium.
–fid *file_id*	Reads the back up filename specified. You assigned this name using **wbak**.
–int (default)	Selects interchange mode. Backup files are restored to disk.
–index	Selects index mode. Backup filenames are listed on standard output; no information is restored to disk.

Catalog Control

–all	Restores or indexes all the objects in the back up file specified. This option is required if you do not use the *pathname* argument to indicate a particular object to be indexed or restored.
–as *pathname1*	Restores the object specified and assign a different disk pathname *pathname1*. This option is valid only when used with the *pathname* argument on the **rbak** command line.
–cr (default)	Specifies create mode. **rbak** does not restore objects if their names already exist on disk. It prints an error message if a name exists on both disk and backup media, and continues.
–r	Specifies replace mode. **rbak** deletes the existing disk object, and replaces it with the object read from backup media.
–force	Forces object deletion if you have owner rights, even if you don't have delete rights.
–du	Deletes when unlocked. If the object to be deleted is locked when **rbak** is invoked, the delete operation is performed when the object is unlocked.
–ms	Specifies merge-source mode. Similar to replace mode. If an object already exists on disk, **rbak** deletes the disk version and restores the backup media version (the source). However, if the object is a directory, **rbak** merges the back up media directory's contents with the disk directory.
–pr *pathname...*	Preserves specified objects on the disk. Multiple pathnames and wildcarding are permitted. If the objects exist on disk, they are not overwritten by backup media versions. This option must be used with the **–ms** option.
–md	Specifies merge–destination mode. Similar to create mode. If an object already exists on disk (the destination) **rbak** does not restore the backup media version, and retains the disk version. However, if the object is a directory **rbak** merges the backup media directory's contents with the disk directory.

Label Control

–sla (default)	Displays the backup media file label on standard output.
–nsla	Does not display the backup media file label.

Listing Control

You may include the **–l** option, or any combination of **–ld**, **–lf**, and **–ll**.

–l	Writes all the file, directory, and link names to standard output.

–ld	Writes all directory names to standard output.
–lf	Writes all filenames to standard output.
–ll	Writes all linknames to standard output.

Backup Device Control

–anys	Forces **rbak** to accept any section of the backup file. When a backup file spans multiple backup media volumes, **rbak** normally begins with the backup media volume containing the backup file's first section, and proceeds to the backup media volume containing the second section, and so on. If you know which backup media volume contains the object you want to restore or index, use this option. This lets **rbak** start at any section of the backup file.
–reo	Forces previous volume to be reopened, and suppress reading of backup media volume label. Use only when backup media has not been repositioned since the last **wbak** or **rbak**.
–dev *d*[*unit*]	Specifies device type and unit number. *d* must be either **m** (for reel–to–reel magnetic tape, **ct** (for cartridge tape), or **f** (for floppy), depending on which drive is being used. *unit* is an integer (0–3). Both are required for reel–to–reel tapes (that is, **–dev m2**). A unit number is not required for floppy disks and cartridge tapes (that is, **–dev f**). If this option is omitted, **rbak** assumes device **m0**. Note: Floppy disk support for this command is limited. In particular, error detection during reads and writes is poor. Do not use this command with floppy disks when the data being placed on the floppy disks is critical and unrecoverable.
–from *filename*	Reads the backup input from a file written by **wbak** using the **–to** option. *filename* specifies the pathname of the file.
–stdin	Specifies the backup input media to be standard input. Used along with I/O redirection, this option is useful for reading files from foreign file systems.
–reten	Retensions the cartridge tape (unwind to the end, then rewind). This can be helpful if you encounter cartridge tape reading errors. Retensioning requires about 1.5 minutes to complete.
–nreten (default)	Do not retension the cartridge tape.

–rewind	Rewinds the cartridge tape after reading or indexing. If this option is omitted, the cartridge tape is left positioned to the next tape file. This option is valid only for the cartridge tape; reel–to–reel tapes are rewound automatically when removed from the drive.

ACL Control

–dacl (default)	Assigns the destination directory's default ACL to the object being restored.
–sacl	Retains the restored object's original ACL.
–pdt	Preserves the object's original date–time modified and date–time used.

EXAMPLES

```
$ rbak –f 1 fred/soup
```

Read **fred/soup** in backup file 1 and restore it to disk. **fred/soup** may be a directory, file, or link.

```
$ rbak –f 1 fred/soup –as //node5/noodle
```

Restore **fred/soup** and place it in **noodle** on node5.

```
$ rbak –dev ct –rewind
```

Rewind the cartridge tape prior to removing it from the tape unit.

```
$ rbak  src –from /fred/bck_out.file
```

Restore the directory **src** to disk. Read the backup input from the file **/fred/bck_out.file**, that should be written by **wbak** using the **–stdout** or **–from** option.

SEE ALSO

wbak(1), rwmt(1)

NAME

rcp – remote file copy

SYNOPSIS

rcp [**–p**] *file1 file2*

rcp [**–p**] [**–r**] *file* ... *directory*

DESCRIPTION

The **rcp** command copies files ɒetween machines. Each *file* or *directory* argument is either a remote file name of the form *rhost:path*, or a local filename containing no colon characters (:) or a slash mark (/) before any colons (:).

By default, **rcp** preserves the mode and owner of *file2* if it already existed; otherwise the mode of the source file modified by the **umask**(2) on the destination host is used.

If *path* is not a full path name, it is interpreted relative to your login directory on *rhost*. A *path* on a remote host may be quoted by using a backslash (\), double quotes ("), or a single quote (´), so that the metacharacters are interpreted remotely.

rcp does not prompt for passwords; your current local user name must exist on *rhost* and allow remote command execution via **remsh**(1C).

rcp handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form *rname@rhost* to use *rname* rather than the current user name on the remote host. The destination hostname may also take the form *host.rname* to support destination machines that are running 4.2BSD versions of **rcp**.

OPTIONS

–r If any of the source files are directories, copy each subtree rooted at that name; in this case the destination must be a directory.

–p Attempt to preserve (duplicate) in copies the modification times and modes of the source files, ignoring the **umask**.

BUGS

Doesn't detect all cases where the target of a copy might be a file in cases where only a directory should be legal.
Is confused by any output generated by commands in a .login, .profile, or .cshrc file on the remote host.

SEE ALSO

cp(1), ftp(1C), remsh(1C), rlogin(1C)

NAME

ed, red – text editor

SYNOPSIS

ed [–s] [–p *string*] [*file*]

red [–s] [–p *string*] [*file*]

DESCRIPTION

ed is the standard text editor. If the *file* argument is given, **ed** simulates an *e* command (see below) on the named file; that is, the file is read into **ed**'s buffer so that it can be edited.

ed operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

red is a restricted version of **ed**. It only allows editing of files in the current directory. It prohibits executing shell commands using the *!shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both **ed** and **red** support the **fspec**(4) formatting capability. After including a format specification as the first line of *file* and invoking **ed** with your terminal in **stty –tabs** or **stty tab3** mode (see **stty**(1)), specified tab stops are automatically used when scanning *file*. For example, if the first line of a file contained:

```
<:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: while inputing text, tab characters when typed are expanded to every eighth column as is the default.

OPTIONS

–s Suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a *!shell command*. Also, see the **WARNING** section at the end of this manual page.

–p Allows you to specify a prompt string. Commands to **ed** have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While **ed** is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line, followed immediately by a carriage return.

ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (*s*, for example) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE.

REGULAR EXPRESSIONS

The following *one-character REs* match a *single* character:

- An ordinary character (*not* one of those discussed below) is a one-character RE that matches itself.
- A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:
  - ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets.
  - ^ (caret or circumflex), which is special at the *beginning of an entire* RE, or immediately follows the left of a pair of square brackets.
  - $ (dollar sign), which is special at the *end* of an entire RE.
  - The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the g command, below.)
- A period (.) is a one-character RE that matches any character except new-line.
- A non-empty string of characters enclosed in square brackets ([ ]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (–) may be used to indicate a range of consecutive ASCII characters; for example, [0–9] is equivalent to [0123456789]. The – loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., [ ]a–f] matches either a right square bracket (]) or one of the letters a through f inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *REs* from one-character REs:

- A one-character RE is a RE that matches whatever the one-character RE matches.
- A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

- A one-character RE followed by \{*m*\}, \{*m,*\}, or \{*m,n*\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m,*\} matches *at least m* occurrences; \{*m,n*\} matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.
- The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.
- A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.
- The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) the sub-expression specified is that beginning with the *n*–th occurrence of \(counting from the left. For example, the expression ^\(.*\)\1$ matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both).

- A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.
- A dollar sign ($) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction ^*entire RE* $ constrains the entire RE to match the entire line.

The null RE (e.g., //) is equivalent to the last RE encountered. See also the last paragraph before **FILES** below.

To understand addressing in **ed** it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

– The character **.** addresses the current line.

– The character $ addresses the last line of the buffer.

– A decimal number *n* addresses the *n*-th line of the buffer.

– *'x* addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.

– A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before **FILES** below.

- A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before **FILES** below.
- An address followed by a plus sign (+) or a minus sign (–) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
- If an address begins with + or –, the addition or subtraction is taken with respect to the current line; e.g, –5 is understood to mean .–5.
- If an address ends with + or –, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and the rule immediately above, the address – refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to –.) Moreover, trailing + and – characters have a cumulative effect, so –– refers to the current line less 2.
- For convenience, a comma (,) stands for the address pair **1,$**, while a semicolon (;) stands for the pair **.,$**.

COMMANDS

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches. The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of **ed** commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n*, and *p* commands.

(.)**a**
<text>

.
: The *a*ppend command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.)c
<text>
.
: The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(. , .)d
: The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*
: The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a shell (**sh**(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also **DIAGNOSTICS** below.

E *file*
: The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file*
: If *file* is given, the *f*ile-name command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1 , $)g/*RE*/*command list*
: In the *g*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted. The . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also **BUGS** and the last paragraph before **FILES** below.

(1,$)G/*RE*/

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an & causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *h*elp command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The *H*elp command causes **ed** to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous ? if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. The address '*x* then addresses this line; . is unchanged.

(.,.)l

The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab, backspace*) are represented by visually mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)m*a*

The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(.,.)n

The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p

The *p*rint command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*. For example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done; however, see **DIAGNOSTICS**, below.

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

($)r *file*

The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (**sh**(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE*/*replacement*/ or
(.,.)s/*RE*/*replacement*/g or
(.,.)s/*RE*/*replacement*/n n = 1-512

The *s*ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator g appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number n appears after the command, only the n th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before **FILES** below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u

The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1 , $)v/*RE*/*command list*

This command is the same as the global command *g* except that the *command list* is executed with . initially set to every line that does *not* match the RE.

(1 , $)V/*RE*/

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the RE.

(1 , $)w *file*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see **umask**(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of c haracters written is typed. If *file* is replaced by **!**, the rest of the line is taken to be a shell (**sh**(1)) command whose standard input is the addressed lines. Such a shellcommand is *not* remembered as the current file name.

($)=

The line number of the addressed line is typed; . is unchanged by this command.

!*shell command*

The remainder of the line after the ! is sent to the UNIX system shell (**sh**(1)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(.+1)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, **ed** prints a ? and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, and 64 characters per file name. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, **ed** discards ASCII NUL characters. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by **ed**.

If a file is not terminated by a new-line character, **ed** adds one and outputs a message explaining what it did.

If the closing delimiter of a RE or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

WARNINGS

The – *option*, although supported in this release for upward compatibility, will no longer be supported in the next major release of the system. Convert shell scripts that use the – *option* to use the –s *option*, instead.

BUGS

A *!* command cannot be subject to a *g* or a *v* command.
The *!* command and the ! escape from the *e*, *r*, and *w* commands cannot be used if the editor is invoked from a restricted shell (see **sh**(1)).
The sequence \n in a RE does not match a new-line character.

Characters are masked to 7 bits on input.

If the editor input is coming from a command file (e.g., ed file < ed-cmd-file), the editor will exit at the first failure.

FILES

/usr/tmp	default directory for temporary work file.
$TMPDIR	if this environmental variable is not null, its value is used in place of **/usr/tmp** as the directory name for the temporary work file.
ed.hup	work is saved here if the terminal is hung up.

DIAGNOSTICS

?	for command errors.
?*file*	for an inaccessible file.
	(use the *h*elp and *H*elp commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, **ed** warns the user if an attempt is made to destroy **ed**'s buffer via the *e* or *q* commands. It prints ? and allows one to continue editing. A second *e* or *q* command at this point will take effect. The –s command-line **option** inhibits this feature.

SEE ALSO

edit(1), ex(1), grep(1), sed(1), sh(1), stty(1), umask(1), vi(1).

fspec(4), regexp(5) in the *SysV Programmer's Reference*.

NAME

regcmp – regular expression compile

SYNOPSIS

regcmp [–] *files*

DESCRIPTION

The **regcmp** command performs a function similar to **regcmp**(3X) and, in most cases, precludes the need for calling **regcmp**(3X) from C programs. This saves on both execution time and program size. The **regcmp** command compiles the regular expressions in *file* and places the output in *file*.**i**. If a simple dash (–) is used in place of a filename, **regcmp** directs the output to *file*.**c**. The format of entries in *file* is a name (C variable), followed by one or more blanks, followed by a regular expression enclosed in double quotes. The output of **regcmp** is C source code. Compiled regular expressions are represented as **extern char** vectors. All *file*.**i** files may thus be *included* in C programs, or *file*.**c** files may be compiled and later loaded. In the C program that uses the **regcmp** output, *regex*(*abc*,*line*) will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

EXAMPLES

```
name  "([A–Za–z][A–Za–z0–9_]*)$0"

telno "\({0,1}([2–9][01][1–9])$0\){0,1} *"
      "([2–9][0–9]{2})$1[ –]{0,1}"
      "([0–9]{4})$2"
```

In the C program that uses the **regcmp** output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

SEE ALSO

regcmp(3X).

NAME

remsh – remote shell

SYNOPSIS

remsh *host* [**–l** *username*] [**–n**] *command*
host [**–l** *username*] [**–n**] *command*

DESCRIPTION

remsh connects to the specified *host,* and executes the specified *command.* **remsh** copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; **remsh** normally terminates when the remote command does.

The remote username used is the same as your local username, unless you specify a different remote name with the –l option.

If you omit *command,* then instead of executing a single command, you will be logged in on the remote host using **rlogin**(1C).

Shell metacharacters which are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine.

Host names are given in the file /etc/**hosts**. Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames. The host names for local machines are also commands in the directory /**usr**/**hosts**; if you put this directory in your search path, then the **remsh** can be omitted.

OPTIONS

–l *username*	Specify a remote *username* different from your local username. This remote name must be equivalent (in the sense of **rlogin**(1C)) to the originating account; no provision is made for specifying a password with a command.
–n	Redirect the input of **remsh** to /**dev**/**null**.

EXAMPLES

The following command appends the remote file **remotefile** to the localfile **localfile.**

```
remsh otherhost cat remotefile >> localfile
```

The command below appends **remotefile** to **otherremotefile.**

```
remsh otherhost cat remotefile ">>" otherremotefile
```

BUGS

If you are using **csh**(1) and put a **remsh**(1C) in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. If no input is desired you should redirect the input of **remsh** to /**dev**/**null** using the **–n** option.

You cannot run an interactive command (such as **rogue**(6) or **vi**(1)); use **rlogin**(1C).

Stop signals stop the local **remsh** process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

FILES

/etc/hosts
/usr/hosts/*

SEE ALSO

rlogin(1C)

NAME

rlogin – remote login

SYNOPSIS

rlogin *rhost* [**–e***c*] [**–8**] [**–L**] [**–l** *username*]
rhost [**–e***c*] [**–8**] [**–L**] [**–l** *username*]

DESCRIPTION

rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost.*

Each host has a file, /etc/**hosts.equiv**, that contains a list of *rhost*s with which it shares account names. (The host names must be the standard names as described in **remsh**(1C).) When you execute **rlogin** as the same user on an equivalent host, you don't need to provide a password. Each user may also have a private equivalence list in a file **.rhosts** in his or her log-in directory. Each line in this file should contain an *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in **login**(1). To avoid some security problems, the **.rhosts** file must be owned by either the remote user or root.

The remote terminal type is the same as your local terminal type (as given in your environment TERM variable). The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. All echoing takes place at the remote site, so that (except for delays) the **rlogin** is transparent. Flow control via CTRL/S and CTRL/Q and flushing of input and output on interrupts are handled properly.

To disconnect from the remote host, use a tilde followed by a period (~.). The tilde is the escape character. Similarly, to suspend the **rlogin** session, use CTRL/Z, the suspend character). By using CTRL/Y (the delayed-suspend character), you can suspend the send portion of the **rlogin**, but allow output from the remote system. Use the **–e** option to specify a different escape character.

OPTIONS

–e*c*	Specify *c* as the escape character to use. There is no space separating **–e** and the argument character.
–8	Allows an eight-bit input data path at all times; otherwise parity bits are stripped except when the remote side's stop and start characters are other than CTRL/S and CTRL/Q.
–L	Allows the **rlogin** session to be run in litout mode.
–l *username*	Specify a different *username*. This is necessary when the originating user is not equivalent to the remote user.

FILES

/usr/hosts/*	For **rhost** version of the command

BUGS

More of the environment should be propagated.

SEE ALSO

remsh(1C)

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [**–f**] [**–i**] *file* ...

rm –r [**–f**] [**–i**] *dirname* ... [*file* ...]

rmdir [**–p**] [**–s**] *dirname* ...

DESCRIPTION

The **rm** command removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with y (for yes), the file is deleted, otherwise the file remains.

Note that if the standard input is not a terminal, the command will operate as if the **–f** option is in effect.

The **rmdir** command removes the named directories, which must be empty.

OPTIONS

The following options apply to the **rm** command:

–f Remove all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory was attempted, this option cannot suppress an error message.

–r Recursively remove any directories and subdirectories in the argument list. The directory will be emptied of files and removed. Normally, you are prompted for removal of any write-protected files that the directory contains. The write-protected files are removed without prompting, however, if the **–f** option is used, or if the standard input is not a terminal and the **–i** option is not used.

If the removal of a non-empty, write-protected directory was attempted, the command will always fail (even if the **–f** option is used), resulting in an error message.

–i With this option, confirmation of removal of any write-protected file occurs interactively. It overrides the **–f** option and remains in effect even if the standard input is not a terminal.

The following options apply to the **rmdir** command:

−p Remove the directory *dirname* and its parent directories which become empty as a result. Print a message on standard output telling whether the whole path is removed or part of the path remains for some reason.

−s Suppress the message printed on standard error when **−p** is in effect.

DIAGNOSTICS

All messages are generally self-explanatory.

It is forbidden to remove the files "." and ".." in order to avoid the consequences of inadvertently doing something like the following:

```
rm −r .*
```

Both **rm** and **rmdir** return exit codes of 0 if all the specified directories are removed successfully. Otherwise, they return a non-zero exit code.

SEE ALSO

unlink(2), rmdir(2).

NAME

mail, rmail – send mail to users or read mail

SYNOPSIS

Sending mail:

mail [**–oswt**] *persons*

rmail [**–oswt**] *persons*

Reading mail:

mail [**–ehpqr**] [**–f** *file*] [**–F** *persons*]

DESCRIPTION

Sending mail:

A *person* is usually a user name recognized by **login**(1). When *persons* are named, **mail** assumes a message is being sent (except in the case of the –F option). It reads from the standard input up to an end-of-file (CTRL/D), or until it reads a line consisting of just a period. When either of those signals is received, **mail** adds the *letter* to the *mailfile* for each *person*. A *letter* is a *message* preceded by a *postmark*. The message is preceded by the sender's name and a *postmark*. A *postmark* consists of one or more 'From' lines followed by a blank line (unless the –s argument was used).

If a letter is found to be undeliverable, it is returned to the sender with diagnostics that indicate the location and nature of the failure. If **mail** is interrupted during input, the file **dead.letter** is saved to allow editing and resending. The **dead.letter** file is recreated every time it is needed, erasing any previous contents.

The **rmail** command only permits the sending of mail; **uucp**(1C) uses **rmail** as a security precaution.

If the local system has the Basic Networking Utilities installed, mail may be sent to a recipient on a remote system. Prefix *person* by the system name and exclamation point. A series of system names separated by exclamation points can be used to direct a letter through an extended network.

Reading Mail:

The **mail** program, unless otherwise influenced by command-line arguments, prints a user's mail messages in last-in, first-out order. For each message, the user is prompted with a question mark (?), and a line is read from the standard input. The following commands are available to determine the disposition of the message:

<newline>, +, or **n**	Go on to next message.
d, or **dp**	Deletes message and go on to next message.
d #	Deletes message number #. Do not go on to next message.
dq	Deletes message and quit **mail**.

h	Displays a window of headers around current message.
h #	Displays header of message number #.
h a	Displays headers of all messages in the user's *mailfile*.
h d	Displays headers of messages scheduled for deletion.
p	Prints current message again.
–	Prints previous message.
a	Prints message that arrived during the *mail* session.
#	Prints message number #.
r [*users*]	Replys to the sender, and other *user(s)*, then deletes the message.
s [*files*]	Saves message in the named *files* (**mbox** is default).
y	Same as save.
u [#]	Undeletes message number # (default is last read).
w [*files*]	Saves message, without its top-most header, in the named *files* (**mbox** is default).
m [*persons*]	Mails the message to the named *persons*.
q, or **ctl-d**	Puts undeleted mail back in the *mailfile* and quits **mail**.
x	Puts all mail back in the *mailfile* unchanged and exits **mail**.
!*command*	Escapes to the shell to do *command*.
?	Prints a command summary.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using **mail**.

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. A "Forwarded by..." message is added to the header. This is especially useful in a multi-machine environment to forward all of a person's mail to a single machine, and to keep the recipient informed if the mail has been forwarded. Installation and removal of forwarding is done with the –F option.

To forward all of one's mail to **systema!user** enter the following:

mail –Fsystema!user

To forward to more than one user, enter this command line:

mail –F"user1,systema!user2,systema!systemb!user3"

Note that when more than one user is specified, the whole list should be enclosed in double quotes so that it may all be interpreted as the operand of the –F option. The list can be up to 1024 bytes; either commas or white space can be used to separate users.

To remove forwarding, enter the following:

mail –F ""

The pair of double quotes is mandatory to set a NULL argument for the –F option.

In order for forwarding to work properly, the *mailfile* should have "mail" as group ID and the group permission should be read-write.

OPTIONS

Sending mail:

–o	Suppresses the address optimization facility.
–s	Suppresses the addition of a <newline> at the top of the letter being sent. See WARNINGS below.
–w	Causes a letter to be sent to a remote user without waiting for the completion of the remote transfer program.
–t	Causes a **To:** line to be added to the letter, showing the intended recipients.

Reading mail:

–e	Causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.
–h	Causes a window of headers to be displayed rather than the latest message. The display is followed by the '?' prompt.
–p	Causes all messages to be printed without prompting for disposition.
–q	Causes **mail** to terminate after interrupts. Normally an interrupt causes only the termination of the message being printed.
–r	Causes messages to be printed in first-in, first-out order.
–f*file*	Causes BI mail to use *file* (e.g., **mbox**) instead of the default *mailfile*.
–F*persons*	Entered into an empty *mailbox*, causes all incoming mail to be forwarded to *persons*.

WARNING

The "Forward to person" feature may result in a loop, if **sys1!userb** forwards to **sys2!userb** and **sys2!userb** forwards to **sys1!userb**. The symptom is a message saying "unbounded...saved mail in dead.letter."

The **–s** option should be used with caution. It allows the text of a message to be interpreted as part of the postmark of the letter, possibly causing confusion to other **mail** programs. To allow compatibility with **mailx**(1), if the first line of the message is "Subject:...", the addition of a <newline> is suppressed whether or not the **–s** option is used.

BUGS

Conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

FILES

/etc/passwd	To identify sender and locate persons
/usr/mail/*user*	Incoming mail for *user*; i.e., the **mailfile**
$HOME/mbox	Saved mail
$MAIL	Variable containing path name of **mailfile**
/tmp/ma*	Temporary file
/usr/mail/*.lock	Lock for mail directory
dead.letter	Unmailable text

SEE ALSO

login(1), mailx(1), write(1).
Managing SysV System Software.

NAME

rmdel – remove a delta from an SCCS file

SYNOPSIS

rmdel −**r***SID files*

DESCRIPTION

The **rmdel** commnd removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the SID specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* (see get(1)) exists for the named SCCS file, the SID specified must *not* appear in any entry of the *p-file*).

The −**r** option is used for specifying the *SID* (SCCS IDentification) level of the delta to be removed.

If a directory is named, **rmdel** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If you specify a simple dash (−) in place of a filename, the standard input is read. Each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

Simply stated, the rules for using **rmdel** are as follows: (1) if you make a delta you can remove it, or (2) if you own the file and directory you can remove a delta.

FILES

x.file	(see **delta**(1))
z.file	(see **delta**(1))

DIAGNOSTICS

Use **help** (1) for explanations.

SEE ALSO

delta(1), get(1), help(1), prs(1), sccsfile(4).

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [–f] [–i] *file* ...

rm –r [–f] [–i] *dirname* ... [*file* ...]

rmdir [–p] [–s] *dirname* ...

DESCRIPTION

The **rm** command removes the entries for one or more files from a directory. If a entry was the last link to the file, the file is destroyed. Removal of a file requires writ permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, the full set of pe missions (in octal) for the file are printed followed by a question mark. This is a promp for confirmation. If the answer begins with y (for yes), the file is deleted, otherwise th file remains.

Note that if the standard input is not a terminal, the command will operate as if the – option is in effect.

The **rmdir** command removes the named directories, which must be empty.

OPTIONS

The following options apply to the **rm** command:

–f Remove all files (whether write-protected or not) in a directory witho prompting the user. In a write-protected directory, however, files a never removed (whatever their permissions are), but no messages a displayed. If the removal of a write-protected directory was attempte this option cannot suppress an error message.

–r Recursively remove any directories and subdirectories in the argume list. The directory will be emptied of files and removed. Normally, yo are prompted for removal of any write-protected files that the directo contains. The write-protected files are removed without promptin however, if the **–f** option is used, or if the standard input is not a termin and the **–i** option is not used.

If the removal of a non-empty, write-protected directory was attempte the command will always fail (even if the **–f** option is used), resulting an error message.

–i With this option, confirmation of removal of any write-protected fi occurs interactively. It overrides the **–f** option and remains in effe even if the standard input is not a terminal.

The following options apply to the **rmdir** command:

−p Remove the directory *dirname* and its parent directories which become empty as a result. Print a message on standard output telling whether the whole path is removed or part of the path remains for some reason.

−s Suppress the message printed on standard error when **−p** is in effect.

DIAGNOSTICS

All messages are generally self-explanatory.

It is forbidden to remove the files "." and ".." in order to avoid the consequences of inadvertently doing something like the following:

```
rm −r .*
```

Both **rm** and **rmdir** return exit codes of 0 if all the specified directories are removed successfully. Otherwise, they return a non-zero exit code.

SEE ALSO

unlink(2), rmdir(2).

NAME

rootnode – change the node to which the root directory refers

SYNOPSIS

rootnode [*nodename*]

DESCRIPTION

rootnode causes / to refer to the node entry directory of *nodename* instead of the current node entry directory. The process must have execute (search) permission for the node entry directory of *nodename*.

Because a new process is created to execute each command, **rootnode** would be ineffective if it were written as a normal command. Therefore, it is an internal command in the Bourne shell, Korn shell, and the C shell.

SEE ALSO

csh(1), sh(1), ksh(1), pwd(1), chdir(2)

NAME

sh, rsh – the standard/restricted Bourne Shell (command programming language)

SYNOPSIS

sh [–acefhiknrstuvx] [–D*name=value* ... **]** [args]
rsh [–acefhiknrstuvx] [–D*name=value* ... **]** [args]

DESCRIPTION

sh is a command programming language that executes commands read from a terminal or a file. **rsh** is a restricted version of the standard command interpreter **sh**; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See ''Invocation'' below for the meaning of arguments to the shell.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *****, **@**, **#**, **?**, **–**, **$**, and **!**.

Commands

A *simple-command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see exec(2)). The *value* of a *simple-command* is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see **signal**(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|**. The standard output of each command but the last is connected by a **pipe**(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **| |**, and optionally terminated by **;** or **&**. Of these four symbols, **;** and **&** have equal precedence, which is lower than that of **&&** and **| |**. The symbols **&&** and **| |** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **&&** (**| |**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a *simple-command* or one of the following. Unless otherwise stated, the value returned by a command is that of the last *simple-command* executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the do *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... esac

A case command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see "File Name Generation") except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [else *list*] **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a sub-shell.

{*list*;}

list is executed in the current (that is, parent) shell.

name () {*list*;}

Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a new line to be ignored.

Command Substitution

The shell reads commands from the string between two grave accents (``) and the standard output from these commands may be used as all or part of a word. Trailing new lines from the standard output are removed. No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (`) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quote

(" ...` ...` ... "), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact. If a backslash is used to escape a newline character (**newline**), both the backslash and the newline are removed (see the later section on "Quoting"). In addition, backslashes used to escape dollar signs (\$) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \, `, ", **newline**, and $ are left intact when the command string is read.

Parameter Substitution

The character $ is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter*}
: The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or @, all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter*:–*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
: If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message ''parameter null or not set'' is printed.

${*parameter*:+*word*}
: If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:–`pwd`}
```

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
–	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last synchronously executed command.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.

PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing under **rsh**.

CDPATH
The search path for the *cd* command.

ENV If this parameter is set, the Shell performs parameter substitution on the value to generate the pathname of the startup script containing commands that the Shell executes every time a new shell is invoked. No error results if the file specified by the ENV parameter doesn't exist or can't be read.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

MAILCHECK
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is *you have mail*.

PS1 Primary prompt string, by default "$ ".

PS2 Secondary prompt string, by default "> ".

IFS Internal field separators, normally **space**, **tab**, and **newline**.

SHACCT
If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed.

SHELL When the shell is invoked, it scans the environment (see ''Environment'' below) for this name. If it is found and 'rsh' is the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH, PS1, PS2, MAILCHECK** and **IFS. HOME** and **MAIL** are set by **login**(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ´´) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Input/Output

A command's input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simple-command* or may precede or follow a *command* and are *not* passed on as arguments to the invoked command. Note that parameter and command substitution occurs before *word* or *digit* is used.

<word Use file *word* as standard input (file descriptor 0).

>word Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

>>word Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[-]word After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file. If, however, – is appended to <<:

1) leading tabs are stripped from *word* before the shell input is read (but after parameter and command substitution is done on *word*),
2) leading tabs are stripped from the shell input as it is read and before each line is compared with *word*, and
3) shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file.

If any character of *word* is quoted (see "Quoting," later), no additional processing is done to the shell input. If no characters of *word* are quoted:

1) parameter and command substitution occurs,
2) (escaped) **newline** is ignored, and
3) \ must be used to quote the characters \, $, and `.

The resulting document becomes the standard input.

<&digit Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using >&**digit**.

<&– The standard input is closed. Similarly for the standard output using >&–.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

```
... 1>xxx 2>&1
```

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

Using the terminology introduced on the first page, under ''Commands,'' if a *command* is composed of several *simple commands*, redirection will be evaluated for the entire *command* before it is evaluated for each *simple command*. That is, the shell evaluates redirection for the entire *list*, then each *pipeline* within the *list*, then each *command* within each *pipeline*, then each *list* within each *command*.

If a command is followed by & the default standard input for the command is the empty file /dev/**null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

File Name Generation

Before a command is executed, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening ``[´´ is a "!" any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > newline space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a backslash (\) or inserting it between a pair of quote marks (´´ or ""). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair **newline** is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (´´), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, "´").

Inside a pair of double quote marks (""), parameter and command substitution occurs and the shell quotes the results to avoid blank interpretation and file name generation. If $* is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces ("$1 $2 ..."); however, if $@ is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces ("$1" "$2" ...). \ quotes the characters \, `, ", and $. The pair \**newline** is removed before parameter and command substitution. If a backslash precedes characters other than \, `, ", $, and newline, then the backslash itself is quoted by the shell.

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

Environment

The *environment* (see **environ**(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd                    and
(export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **–k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c
set –k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via exec(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (**:**). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign, between two colon delimiters anywhere in the path list, or at the end of the path list. If the command name contains a **/** the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
: Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

continue [*n*]
: Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.

cd [*arg*]
: Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (**:**). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command may not be executed by **rsh**.

echo [*arg* ...]
: Echo arguments. See **echo**(1) for usage and description.

eval [*arg* ...]
: The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
: The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
: Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]
: The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are *not* exported.

getopts Use in shell scripts to support command syntax standards (see **intro**(1)); it parses positional parameters and checks for legal options. See *getopts*(1) for usage and description.

hash [**-r**] [*name* ...]
: For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given,

information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]
: Equivalent to **exec newgrp** *arg* .... See **newgrp**(1) for usage and description.

pwd Print the current working directory. See **pwd**(1) for usage and description.

read [*name* ...]
: One line is read from the standard input and, using the internal field separator, **IFS** (normally space or tab), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, etc., with left-over words assigned to the last *name*. Lines can be continued using **newline**. Characters other than **newline** can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
: The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
: Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**—aefhkntuvx** [*arg* ...]]

–a	Mark variables which are modified or created for export.
–e	Exit immediately if a command exits with a non-zero exit status.
–f	Disable file name generation
–h	Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
–k	All keyword arguments are placed in the environment for a command, not just those that precede the command name.
–n	Read commands but do not execute them.
–t	Exit after reading and executing one command.
–u	Treat unset variables as an error when substituting.
–v	Print shell input lines as they are read.

−x Print commands and their arguments as they are executed.

— Do not change any of the flags; useful in setting **$1** to −.

Using + rather than − causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in $−. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given the values of all names are printed.

shift [*n*]
The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

test
Evaluate conditional expressions. See **test**(1) for usage and description.

times
Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [*n*]
Impose a size limit of *n* blocks on files written by the shell and its child processes (files of any size may be read). If *n* is omitted, the current limit is printed. You may lower your own ulimit, but only a super-user (see **su**(1M)) can raise a ulimit.

umask [*nnn*]
The user file-creation mask is set to *nnn* (see **umask**(1)). If *nnn* is omitted, the current value of the mask is printed.

unset [*name* ...]
For each *name*, remove the corresponding variable or function. The variables **PATH, PS1, PS2, MAILCHECK** and **IFS** cannot be unset.

wait [*n*]
Wait for your background process whose process id is *n* and report its

termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

Invocation

If the shell is invoked through exec(2) and the first character of argument zero is –, commands are initially read from */etc/profile* and from *$HOME/.profile*, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as */bin/sh*. The flags below are interpreted by the shell on invocation only; Note that unless the –c or –s flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

–c *string*	If the –c flag is present commands are read from *string*.
–s	If the –s flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.
–i	If the –i flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.
–r	If the –r flag is present the shell is a restricted shell.
–D*name=value*	Use the –D option to specify a parameter *name*, that will be set to *value*, then passed into the shell's environment. This SysV option is useful for tailoring the environment of a shell invoked from a program that is not another shell (such as the Display Manager). If you set the ENV parameter using this option, the startup script it specifies will be run. Any number of –D options can be specified.

The remaining flags and arguments are described under the set command above.

rsh Only

rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of **rsh** are identical to those of **sh**, except that the following are disallowed:

changing directory (see **cd**(1)),
setting the value of **$PATH,**
specifying path or command names containing /,
redirecting output (> and >>).

The restrictions above are enforced after *.profile* is interpreted.

A restricted shell can be invoked in one of the following ways: (1) **rsh** is the file name part of the last entry in the */etc/passwd* file (see **passwd**(4)); (2) the environment variable **SHELL** exists and **rsh** is the file name part of its value; (3) the shell is invoked and **rsh** is the file name part of argument 0; (4) the shell is invoke with the **−r** option.

When a command to be executed is found to be a shell procedure, **rsh** invokes **sh** to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the *.profile* (see **profile**(4)) has complete control over user actions by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by a restricted shell. Some systems also provide a restricted editor, *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

CAVEATS

Words used for filenames in input/output redirection are not interpreted for filename generation (see ''File Name Generation,'' above). For example, **cat file1 >a*** will create a file named **a***.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message *cannot fork, too many processes*, try using the *wait*(1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

BUGS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For **wait n**, if *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cd(1), echo(1), env(1), getopts(1), intro(1), login(1), newgrp(1), pwd(1), test(1), umask(1), wait(1).
dup(2), exec(2), fork(2), pipe(2), profile(4), signal(2), ulimit(2) in the *SysV Programmer's Reference*.

NAME

ruptime – show host status of local machines

SYNOPSIS

ruptime [**–a**] [**–r**] [**–t**] [**–u**]

DESCRIPTION

The **ruptime** command gives a status line for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 11 minutes are shown as being down. Users idle an hour or more are not counted unless the **–a** flag is given. Normally, the listing is sorted by host name.

OPTIONS

–a	Count the users who've been idle for an hour or more.
–r	Reverse the sort order.
–t	Sort the listing by uptime.
–u	Sort the listing by number of users.

FILES

/usr/spool/rwho/whod.* data files

SEE ALSO

rwho(1C)

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [–a]

DESCRIPTION

The **rwho** command produces output similar to **who**, but for all machines on the local network. If no report has been received from a machine for five minutes, **rwho** assumes the machine is down, and does not report users last known to be logged into that machine.

If a user hasn't typed to the system for a minute or more, **rwho** reports this idle time. If a user hasn't typed to the system for an hour or more, the user is omitted from the output of **rwho**, unless you specified the –a flag.

BUGS

The **rwho** command is unwieldy when the number of machines on the local net is large.

FILES

/usr/spool/rwho/whod.*	Information about other machines.

SEE ALSO

ruptime(1C), rwhod(1M),
Managing SysV System Software.

NAME

rwmt – read/write foreign magtapes

SYNOPSIS

rwmt [*option*]... [**–p**] {**–r**|**–w**|**–i**|**–l**} [*pathname*]...

DESCRIPTION

rwmt reads tapes from non–Domain installations and writes tapes that can be read by non–Domain installations. **rwmt** can read and write unlabeled tapes, as well as ANSI level 1–4 labeled tapes.

pathname (optional) Specify the name of file to be read from or written to tape. This argument is valid only with the **–r** and **–w** mode-control options (below). Multiple pathnames are permitted. Wildcarding is permitted for write (–w) operations only.

Default if omitted: read pathnames from standard input

OPTIONS

Mode control

You must specify one of the following mode-control options. If you omit this option, **rwmt** prompts you for it. The **–p** option tells **rwmt** to prompt for all necessary options.

–l[abel] Write ANSI X3.27–1978 volume label on a tape. This option causes **rwmt** to write an ANSI volume label and dummy file on the magtape volume. You may specify an optional owner and volume ID, which are stored in the volume label. (see **–vid** and **–own** below. This is the way to initialize a labeled tape; if any information existed on the tape, it is erased by this labeling operation.

If you are labeling a tape, you can also use the following two options.

–vid *vol_id* Specifies a 1–6 character volume ID for use when labeling a volume. This option is valid only when used with the **–l** mode-control option (above). The default volume ID is ' ' (blank).

–own *owner_id* Specifies a 1–14 character owner ID for use when labeling a volume. This option is valid only when used with the **–l** mode-control option (above). The default owner ID is ' ' (blank).

–i[ndex] Lists objects on an ANSI–labeled physical tape volume. **–index** produces a listing of all files or file sections on an ANSI–labeled physical tape volume. The contents of the physical volume (VOL1) label and all file header labels are written to standard output.

−w[rite]
Specifies one or more disk files (*pathname* argument) to be written to tape. The default format is ANSI labeled, ASCII, fixed–length records of 80 bytes each, and 80–byte blocks. If desired, any of these parameters can be changed using the options described below. If more than one pathname is specified, the disk files are written to sequential tape files. Tapes written by **rwmt** are always in accordance with ANSI level 4 format.
Before writing a labeled file, the tape volume itself must be labeled with the **−label** mode-control option (above).

−r[ead]
Specifies one or more tape files to be read from tape and stored on disk. **read** reads one or more tape files and writes them to disk using the specified pathnames (*pathname* argument). The default tape file format is the same as that for the **write** option. If the tape is labeled under ANSI level 2, 3, or 4, the file format (block length, record length, and record format) is read from the tape. If the tape is unlabeled, or labeled with ANSI level 1, you must specify the tape format using the options below. If more than one pathname is specified, adjacent tape files are read and stored under the specified pathnames.

Label Control

−ansi (default)
Specifies that the tape is labeled in conformance to ANSI X3.27−1978, level 1, 2, 3, or 4.

−unlab
Specifies that the tape is unlabeled. Data spanning physical volumes is not supported on unlabeled tapes.

−asc (default)
Specifies that all tape file contents are in ASCII characters.

−ebc
Specifies that all tape file contents (except labels) are in EBCDIC characters.

−raw
Specifies that all tape file data is to be treated in raw form.

−npar (default)
Specifies no disturbance of parity bits when reading or writing data.

−par
Specifies that parity bits should be forced off when reading data from tape and forced on when writing data to tape.

−rl *reclen*
Specifies the maximum length, in bytes, of a record in the tape file. This option is valid only when used with either the −r or the −w mode-control options (above). It is unnecessary when reading an ANSI level 2, 3, or 4 file. The default record length is 80 bytes.

–bl *blocklen*
Specifies the length, in bytes, of a physical tape block. This option is valid only when used with either the **–r** or the **–w** mode-control options (above). It is unnecessary when reading an ANSI level 2, 3, or 4 file. The default block length is 80 bytes.

–bf *blockfac*
Specifies a blocking factor — the number of records to store into or read from a physical tape block. This is an alternative to the **–bl** option, since the record length multiplied by the blocking factor yields the block length. This option is valid only when used with either the **–r** or **–w** mode-control options (above). Using this option is meaningful only if your tape has fixed–length records. This option is unnecessary when reading an ANSI level 2, 3, or 4 file. The default blocking factor is 1.

–rf *format*
Specifies record format. Valid values for *format* are **f** (fixed–length records and blocks); **d** (variable–length records (this is ANSI 'D' format)); s (spanned records); or **u** (undefined record format). The default format is **f**. Note that if you are writing a cartridge tape, only 512 byte blocks may be written; **d**, s, and **u** formats are not supported.

Tape File Identifiers

–fid *file_id*
Specifies a 1–17 character file ID to be written in the file header label for use when writing a file to a labeled volume. This option is valid only when used with the –w mode-control option (above). If this option is omitted, the name of the file being written is used.

–f [*position*]
Specifies the file position for **–r** or –w operations. Valid values for *position* are **cur**, **end**, or a nonzero integer value. A position of **cur** selects the current tape position; the tape must have been previously read or written by **rwmt** and its position must not have been disturbed. This option is valid only when used with either the **–r** or the –w mode-control options (above).

A position of **end** selects the end of the tape file set. This option is valid only when used with the –w mode-control option, and causes **rwmt** to append the specified disk file (*pathname* argument) to the very end of the tape file set.

A position specified by a nonzero integer value selects the file at that absolute position in the tape volume. This option is valid only when used with either the **–r** or –w mode-control options (above). If multiple *pathname* arguments are supplied, the value of *position* is incremented by one after each file has been read or written.

The default value for *position* is 1.

Backup Device Control

–dev d[*unit*]
: Specifies device type and unit number. *d* must be either **m** (for reel–to–reel magnetic tape), **ct** (for cartridge tape), or **f** (for floppy), depending on which drive is being used. *unit* is an integer (0–3). Both are required for reel–to–reel tapes (that is, **–dev** m2). A unit number is not required for floppy disks and cartridge tapes (that is, **–dev f**). If this option is omitted, **rbak** assumes device **m0**.

–nobs
: Specifies that byte swapping should not be done in software. This operation is valid for magnetic tapes only. On the multibus data gets byte swapped. **rwmt** does byte swapping in software so that the tape gets written out in the correct machine order. **wbak** and **rbak** do not do byte swapping in software, as a result the two swaps done by the multibus cancel out. This option is useful in writing to a magnetic tape an intermediate file to which **wbak** output has been directed. Byte swapping should not be done by **rwmt** if the intermediate file written by **wbak** is now written raw to the magnetic tape using **rwmt**.

–reten
: Retensions the cartridge tape (unwind to the end, then rewind). This can be helpful if you have encountered cartridge-tape reading errors. Retensioning requires about 1.5 minutes to complete.

–nreten (default)
: Does not retension the cartridge tape.

Miscellaneous Control Options

–sbin
: Causes all stream files written to contain the binary attribute (normally, output stream files contain the ASCII attribute).

–p
: Causes **rwmt** to prompt for all unspecified parameters.

EXAMPLES

Initialize a tape with the given owner and volume ID.

```
$ rwmt –label –own "R and D" –vid "demo"
List the files on the tape.
```

```
$ rwmt –index
```

```
Volume label:
   Volume ID: "DEMO  "    Owner ID: "R AND D        "    Access: " "
```

```
File/Section      File ID          Cr Date     Acc   RF     RL     BL
  1     1      CMF_EXAMPLE         83/02/17          D      200    2048
  2     1      CMT_EXAMPLE         83/02/17          D      200    2048
  3     1      CPBOOT_EXAMPLE      83/02/17          D      200    2048
  4     1      CPF_EXAMPLE         83/02/17          D      200    2048
  5     1      CPT_EXAMPLE         83/02/17          D      200    2048

End of file set.
$
```

Copy tape file 3 to a disk file named **cpboot_example.tape.**

```
$ rwmt –r cpboot_example.tape –f 3
4 records read from tape file  3 into
  "cpboot_example.tape".
$
```

rwmt permits a tape file to be read in "raw" mode. In this mode, each block read from the tape is written into one record in a stream file, unmodified by the program. To read a file in "raw" mode, you should specify the maximum record size using the **–rl** argument. If you do not, the default value of 80 bytes is used, and any records longer than that are truncated. Also, undefined record format should be used. For example

```
$  rwmt –r –f 1 –rf u –raw –rl 512 rawfile
```

reads tape file number 1 into **rawfile**, with a maximum record length of 512 bytes.

Files may be written in the same manner:

```
$ rwmt –w –f 1 –rf u –raw –rl 512 rawfile
```

The file **//backup/tmp1** is written out to the magnetic tape in "raw" mode. The record length is specified to be 8k and no byte swapping is done in software. This is useful for writing out an intermediate file to which **wbak** has written its output. Note that all tapes written by **rwmt** must have a ANSI standard volume label for **rbak** to be able to read the tape

```
rwmt –w –f 1 –raw –rl 8192 –nobs //backup/tmp1 –ansi
```

If **rwmt** writes a file with **–nobs** option, you should use **–nobs** option to read it using **rwmt.**

SEE ALSO

rbak(1), wbak(1)

NAME

sact – print current SCCS file editing activity

SYNOPSIS

sact *files*

DESCRIPTION

The **sact** command informs you of any impending deltas to a named SCCS file. This situation occurs when **get**(1) with the **–e** option has been previously executed without a subsequent execution of **delta**(1). If a directory is named on the command line, **sact** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If you specify a simple dash (–) in place of a filename, **sact** reads the standard input, taking each line as the name of an SCCS file to be processed. The output for each named file consists of five fields separated by spaces:

Field 1	Specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	Specifies the SID for the new delta to be created.
Field 3	Contains the logname of the user who will make the delta (i.e., executed a **get** for editing).
Field 4	Contains the date that **get –e** was executed.
Field 5	Contains the time that **get –e** was executed.

DIAGNOSTICS

Use **help**(1) for explanations.

SEE ALSO

delta(1), get(1), unget(1).

NAME

sccs – front end for the SCCS subsystem

SYNOPSIS

sccs [**–r**] [**–d***path*] [**–p***path*] *command* [*flags*] [*arg* ...]

DESCRIPTION

The sccs command is a front end to the Source Code Control System (SCCS) programs that helps them mesh more cleanly with the rest of UNIX. It also includes the capability to run ''set user ID'' to another user to provide additional protection.

Basically, sccs runs the *command* with the specified *flags* and *args*. Each argument is normally modified to be prefixed with SCCS/s.

Flags to be interpreted by the sccs program must appear before the *command* argument. Flags to be passed to the actual SCCS program must come after the *command* argument. These flags are specific to the command and are discussed in the documentation for that command. The SEE ALSO section lists the standard SCCS commands that are documented in section one of this manual. For more information about the standard SCCS commands, see the documentation for that command.

Besides the usual SCCS commands, several pseudo commands can be issued. These pseudo commands are described in the following list.

edit	Equivalent to get –e
delget	Perform a delta on the named files and then get new versions. The new versions will have ID keywords expanded, and will not be editable. The **–m**, **–p**, **–r**, –s, and –y flags will be passed to delta, and the **–b**, **–c**, **–e**, **–i**, **–k**, **–l**, –s, and –x flags will be passed to get.
deledit	Equivalent to **delget** except that the **get** phase includes the **–e** flag. This option is useful for making a checkpoint of your current editing phase. The same flags will be passed to delta as described above, and all the flags listed for **get** above, except **–e** and **–k**, are passed to **edit.**
create	Creates an SCCS file, taking the initial contents from the file of the same name. Any flags to **admin** are accepted. If the creation is successful, the files are renamed with a comma on the front. These should be removed when you are convinced that the SCCS files have been created successfully.

fix Must be followed by a **−r** flag. This command essentially removes the named delta, but leaves you with a copy of the delta with the changes that were in it. It is useful for fixing small compiler bugs, etc. Since it doesn't leave audit trails, it should be used carefully.

clean This routine removes everything from the current directory that can be recreated from SCCS files. It will not remove any files being edited. If the **−b** flag is given, branches are ignored in the determination of whether they are being edited; this is dangerous if you are keeping the branches in the same directory.

unedit This is the opposite of an **edit** or a get −e. It should be used with extreme caution, since any changes you made since the get will be irretrievably lost.

info Gives a listing of all files being edited. If the **−b** flag is given, branches (i.e., SID's with two or fewer components) are ignored. If the **−u** flag is given (with an optional argument) then only files being edited by you (or the named user) are listed.

check Like **info** except that nothing is printed if nothing is being edited, and a non-zero exit status is returned if anything is being edited. The intent is to have this included in an ''install'' entry in a makefile to insure that everything is included into the SCCS file before a version is installed.

tell Gives a newline-separated list of the files being edited on the standard output. Takes the **−b** and **−u** flags like **info** and **check**.

diffs Gives a **diff** listing between the current version of the program(s) you have out for editing and the versions in SCCS format. The **−r**, **−c**, **−i**, **−x**, and **−t** flags are passed to get; the **−l**, **−s**, **−e**, **−f**, **−h**, and **−b** options are passed to **diff.** The −C flag is passed to **diff** as **−c.**

print This command prints out verbose information about the named files.

OPTIONS

−r Runs sccs as the real user rather than as whatever effective user sccs is ''set user ID'' to.

−d Gives a root directory for the SCCS files. The default is the current directory.

−p Defines the pathname of the directory in which the SCCS files will be found; SCCS is the default.

The **−p** flag differs from the **−d** flag in that the **−d** argument is prepended to the entire pathname and the **−p** argument is inserted before the final component of the pathname.

For example:

```
sccs -d/x -py get a/b
```

will convert to:

```
get /x/a/y/s.b
```

The intent here is to create aliases such as:

```
alias syssccs sccs -d/usr/src
```

which will be used as:

```
syssccs get cmd/who.c
```

If the environment variable PROJECT is set, its value is used to determine the **–d flag.** If it begins with a slash, it is taken directly; otherwise, the home directory of a user of that name is examined for a subdirectory **src** or **source**. If such a directory is found, it is used.

Certain commands (such as **admin**) cannot be run ''set user ID'' by all users, since this would allow anyone to change the authorizations. These commands are always run as the real user.

EXAMPLES

To get a file for editing, edit it, and produce a new delta:

```
sccs get –e file.c
ex file.c
sccs delta file.c
```

To get a file from another directory:

```
sccs –p/usr/src/sccs/s. get cc.c
```

or

```
sccs get /usr/src/sccs/s.cc.c
```

To make a delta of a large number of files in the current directory:

```
sccs delta *.c
```

To get a list of files being edited that are not on branches:

```
sccs info –b
```

To delta everything being edited by you:

```
sccs delta `sccs tell –u`
```

In a makefile, to get source files from an SCCS file if it does not already exist:

```
SRCS = <list of source files>
$(SRCS):
        sccs get $(REL) $@
```

NOTES

It should be able to take directory arguments on pseudo commands like the SCCS commands do.

SEE ALSO

admin(1), comb(1), delta(1), get(1), help(1), rmdel(1), sact(1), sccsdiff(1), what(1); *Domain/OS Programming Environment Reference*

NAME

sccsdiff – compare two versions of an SCCS file

SYNOPSIS

sccsdiff **–r** *SID1* **–r** *SID2* [–p] [–s *n*] *files*

DESCRIPTION

The **sccsdiff** command compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

OPTIONS

–r*SID?*	Use *SID1* or *SID2* to specify the deltas of an SCCS file that are to be compared. Versions are passed to **bdiff**(1) in the order given.
–p	Pipe output for each file through **pr**(1).
–s*n*	Specify *n* as the file segment size that *bdiff* will pass to *diff*(1). This is useful when **diff** fails due to a high system load.

FILES

/tmp/get?????	Temporary files

DIAGNOSTICS

file : No differences	The two versions are the same.

Use *help*(1) for explanations.

SEE ALSO

get(1), bdiff(1), help(1), pr(1).

NAME

scrattr – screen attributes

SYNOPSIS

scrattr [**–avcpxy**]

DESCRIPTION

Without any options, **scrattr** lists the X and Y dimensions of the display in pixels. Screen attributes are always listed in the same order: X coordinates, Y coordinates, number of planes, number of primary colors. This order is independant of the order in which the options are specified.

OPTIONS

–a Display all attributes.

–v Print a verbose description of each field, with the attributes on separate lines. (Without the –v option, attributes appear on the same line separated by tabs.) If any options other than –v are specified, only that combination of attributes will be displayed, and always in the canonical order given above.

–c Display the number of primary colors on the display.

–p Display the number of bit planes on the display.

–x Display the X dimension of the display in pixels.

–y Display the Y dimension of the display in pixels.

SEE ALSO

stty(1), tabs(1), tset(1), tty(1)

NAME

scrto – set/show screen timeout

SYNOPSIS

scrto [–none] [*n*]

DESCRIPTION

scrto sets or displays the number of minutes the system waits before it shuts off the display screen. It begins counting minutes after the last input event or window configuration change.

By default, the system waits 15 minutes before it shuts off the display. Domain/OS turns the display back on whenever it receives an input event from the keyboard or mouse or whenever the DM creates, pops, moves, or resizes a window.

n (optional) Set the number of Domain/OS minutes for to wait before it shuts off the display.

Default if omitted: display current timeout setting

OPTIONS

–none Disables automatic timeout; never turn off the display.

EXAMPLES

Shows initial setting.

```
$ scrto
The screen timeout is set to 15 minutes
$
```

Sets delay to 10 min.

```
$ scrto 10
$
```

NAME

sdiff – side-by-side difference program

SYNOPSIS

sdiff [*options* ...] *file1 file2*

DESCRIPTION

The **sdiff** command uses the output of **diff**(1) to produce a side-by-side listing of twc files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a less-than sign (<) in the gutter i the line only exists in *file1*, a greater-than sign (>) in the gutter if the line only exists ir *file2*, and a pipe character (|) for lines that are different.

For example:

```
x    |    y
a         a
b    <
c    <
d         d
     >    c
```

OPTIONS

–w *n* Uses the next argument, *n*, as the width of the output line. The defaul line length is 130 characters.

–l Only prints the left side of any lines that are identical.

–s Does not print identical lines.

–o *output* Uses the next argument, *output*, as the name of a third file that is create as a user-controlled merging of *file1* and *file2*. Identical lines of *file* and *file2* are copied to *output*. Sets of differences, as produced b **diff**(1), are printed; where a set of differences share a common gutte character. After printing each set of differences, **sdiff** prompts you wit a percent character (%) and waits for one of the following user-type commands:

l	Appends the left column to the output file
r	Appensd the right column to the output file
s	Turns on silent mode; do not print identical lines
v	Turns off silent mode

e l	Calls the editor with the left column
e r	Calls the editor with the right column
e b	Calls the editor with the concatenation of left and right
e	Calls the editor with a zero length file
q	Exits from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1).

NAME

sed – stream editor

SYNOPSIS

sed [**–n**] [**–e** *script*] [**–f** *sfile*] [*files*]

DESCRIPTION

The **sed** command copies the named *files* (standard input default) to the standard output, edited according to a script of commands.

A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, **sed** cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **–n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of **ed**(1) modified thus:

- In a context address, the construction \*?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address \xabc\xdefx, the second x stands for itself, so that the regular expression is **abcxdef**.
- The escape sequence **\n** matches a newline *embedded* in the pattern space.
- A period **.** matches any character except the *terminal* newline of the pattern space.
- A command line with no addresses selects every pattern space.
- A command line with one address selects each pattern space that matches the address.
- A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (See the Functions section below).

OPTIONS

–e *script*

Edit according to *script*. If there is just one **–e** option and no **–f** options, the flag **–e** may be omitted.

−f *sfile*	Take the script from file *sfile*; these options accumulate.
−n	Suppress the default output.

FUNCTIONS

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with \ to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) **a\** *text*	Append. Place *text* on the output before reading the next input line.
(2) **b** *label*	Branch to the : command bearing the *label*. If *label* is empty, branch to the end of the script.
(2) **c\** *text*	Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.
(2) **d**	Delete the pattern space. Start the next cycle.
(2) **D**	Delete the initial segment of the pattern space through the first newline. Start the next cycle.
(2) **g**	Replace the contents of the pattern space by the contents of the hold space.
(2) **G**	Append the contents of the hold space to the pattern space.
(2) **h**	Replace the contents of the hold space by the contents of the pattern space.
(2) **H**	Append the contents of the pattern space to the hold space.
(1) **i\** *text*	Insert. Place *text* on the standard output.
(2) **l**	List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.
(2) **n**	Copy the pattern space to the standard output. Replace the pattern space with the next line of input.
(2) **N**	Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)
(2) **p**	Print. Copy the pattern space to the standard output.

(2) **P** Copy the initial segment of the pattern space through the first newline to the standard output.

(1) **q** Quit. Branch to the end of the script. Do not start a new cycle.

(2) **r** *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) s/*regular expression*/*replacement*/*flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see **ed**(1). *Flags* is zero or more of:

n	n= 1 - 512. Substitute for just the n th occurrence of the *regular expression.*
g	Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
p	Print the pattern space if a replacement was made.
w *wfile*	Write. Append the pattern space to *wfile* if a replacement was made.

(2) **t** *label* Test. Branch to the colon (:) command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.

(2) **w** *wfile* Write. Append the pattern space to *wfile*.

(2) **x** Exchange the contents of the pattern and hold spaces.

(2) y/*string1*/*string2*/ Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)**!** *function* Don't. Apply the *function* (or group, if *function* is a left brace ({) only to lines *not* selected by the address(es).

(0)**:** *label* This command does nothing; it bears a *label* for **b** and **t** commands to branch to.

(1)**=** Place the current line number on the standard output as a line.

(2)**{** Execute the following commands through a matching right brace (}) only when the pattern space is selected.

(0) An empty command is ignored.

(0)# If a pound sign (#) appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the pound sign is an 'n', then the default output will be suppressed. The rest of the line after #n is also ignored. A script file must contain at least one non-comment line.

SEE ALSO

awk(1), ed(1), grep(1).

NAME

sh – the Bourne shell command language

SYNOPSIS

sh [**–ceiknrstuvx**] [**–D***name=value* ...] [*arg*] ...

DESCRIPTION

sh is a command programming language that executes commands read from a terminal or a file. A *simple command* is a sequence of nonblank *words* separated by blanks. A blank is a *tab* or a *space*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0; refer to **execve**(2) for further details. The *value* of a simple command is its exit status if it terminates normally, or 200+*status* if it terminates abnormally. See **sigvec(2)** for a list of status values.

A *pipeline* is a sequence of one or more *commands* separated by **|**. The standard output of each command but the last is connected by a **pipe**(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&** or **||** and optionally terminated by **;** or **&**. **;** and **&** have equal precedence which is lower than that of **&&** and **||**, **&&** and **||** also have equal precedence. A semicolon causes sequential execution; an ampersand causes the preceding pipeline to be executed without waiting for it to finish. The symbol **&&** (**||**) causes the *list* following to be executed only if the preceding *pipeline* returns a zero (non zero) value. Newlines may appear in a *list,* instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. The value returned by a command is that of the last simple-command executed in the command.

COMMANDS

for *name* [**in** *word* ...] **do** *list* **done**

Set *name* to the next word in the **for** word list each time a **for** command is executed. If **in** *word* ... is omitted, **in $@** is assumed. Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... esac

Execute the *list* associated with the first pattern that matches *word.* The form of the patterns is the same as that used for filename generation.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

Execute the *list* following **if**, and if it returns zero, execute the *list* following **then**. Otherwise, execute the *list* following **elif**, and if its value is zero, execute the *list* following **then**. Failing that, execute the **else** *list*.

while *list* [**do** *list*] **done**

Repeatedly execute the **while** *list,* and if its value is zero, execute the **do** *list;* otherwise, terminate the loop. The value returned by a **while** command is that of the last executed command in the **do** *list.* To negate the loop termination test, use **until** in place of **while.**

(*list*) Execute *list* in a sub-shell.

{ *list* } Simply execute *list.*

sh recognizes the following words only when they are the first word of a command, and only when they are not quoted:

if then else elif fi case in esac for while until do done { }

COMMAND SUBSTITUTION

Use the standard output from a command, enclosed in a pair of back quotes (\` \`), as part or all of a word. Trailing newlines are removed.

PARAMETER SUBSTITUTION

A *parameter* is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the following characters: *** @ # ? – $!**. The character **$** is used to introduce substitutable parameters. Positional parameters may be assigned values by **set**. Variables may be set by writing

name=value [*name=value*] ..

$ {*parameter*}

Substitute the value, if any, of the parameter. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is a digit, it is a positional parameter. If *parameter* is ***** or **@** then all the positional parameters, starting with **$1**, are substituted separated by spaces. **$0** is set from argument zero when the shell is invoked.

$ {*parameter–word*}

If *parameter* is set, substitute its value; otherwise, substitute *word.*

$ {*parameter=word*}

If *parameter* is not set, set it to *word;* then, substitute the value of the parameter. You may not assign positional parameters in this manner.

$ {*parameter ? word*}

If *parameter* is set, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, a standard message is printed.

$ {*parameter+word*}

If *parameter* is set, substitute *word;* otherwise, substitute nothing.

In the above, **sh** does not evaluate *word* unless it is to be used as the substituted string. Thus, for example, **echo ${d–´pwd´}** only executes **pwd** if *d* is unset.

The following *parameters* are automatically set by the shell:

#	Number of positional parameters in decimal.
–	Options supplied to the shell on invocation or by set.
?	Value returned by the last executed command in decimal.
$	Process number of this shell.
!	Process number of the last background command invoked.

The following *parameters* are used but not set by the shell:

HOME	Default argument (home directory) for the **cd** command.
PATH	The search path for commands (see EXECUTION).
ENV	If this parameter is set, the shell performs parameter substitution on the value to generate the pathname of the startup script containing commands that the shell executes every time a new shell is invoked. No error results if the file specified by the ENV parameter doesn't exist or can't be read.
MAIL	If this variable is set to the name of a mail file, the shell informs you of the arrival of mail in the specified file.
PS1	Primary prompt string; this is a dollar sign ($) by default.
PS2	Secondary prompt string; this is a greater-than sign (>) by default.
IFS	Internal field separators; these usually include *space*, *tab*, and *new-line*. **IFS** is ignored if **sh** is running as root or if the effective user id differs from the real user id.

BLANK INTERPRETATION

After parameter and command substitution, **sh** scans the subsequent results for internal field separator characters (those found in $IFS), and then splits its output into distinct arguments where such characters are found. Explicit null arguments ("" or ´´) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

FILENAME GENERATION

Following substitution, **sh** scans each command word for the characters *, ? and [. If one of these characters appears, the word is regarded as a pattern. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The character **.** at the start of a filename or immediately following a /, and the character /, must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the characters enclosed. A pair of characters separated by – matches any character lexically between the pair.

QUOTING

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | < > **newline space tab**

You may quote a character by preceding it with a \. **sh** ignores a **newline.** All characters enclosed between a pair of single quotation marks (´´), except a single quote, are quoted. Parameter and command substitution occurs inside double quotes (""). A \ quotes the following characters: \ ´ " and $.

A "$*" is equivalent to "$1 $2 ...", while "$@" is equivalent to "$1" "$2"....

PROMPTING

When used interactively, **sh** prompts with the value of **PS1** before reading a command. If, at any time, you type a newline and need further input to complete a command, **sh** issues the secondary prompt (**$PS2**).

INPUT/OUTPUT

Before you execute a command, you may redirect its output by using a special notation interpreted by the shell. The following may appear anywhere in a simple command or may precede or follow a *command,* and are not passed on to the invoked command. Substitution occurs before *word* or *digit* is used.

<*word* Use file *word* as standard input (file descriptor 0).

>*word* Use file *word* as standard output (file descriptor 1). If the file does not exist, create it; otherwise, truncate it to zero length.

>>*word* Use file *word* as standard output. If the file exists, append output (by seeking to the end); otherwise, create the file.

<<*word* Read the shell input up to a line the same as *word* or end-of-file. Make the resulting document the standard input. If any character of *word* is quoted, place no interpretation upon the characters of the document. Otherwise, do parameter and command substitution, ignore **newline** and use \ to quote the following characters: \ $ ´ and the first character of *word.*

<& *digit*

Duplicate the standard input from file descriptor *digit.* This works similarly for the standard output using >. Refer to **dup**(2) for further details.

<&– Close the standard input. Perform the same function on the standard output, using >.

If one of the above is preceded by a digit, the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example,

```
... 2>&1
```

creates file descriptor 2 to be a duplicate of file descriptor 1.

If a command is followed by & the default standard input for the command is the empty file (**/dev/null**). Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

ENVIRONMENT

The environment is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. Refer to **execve**(2) and **environ**(7). The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a *parameter* for each name found, giving it the corresponding value. Executed commands inherit the same environment. If you modify the values of these parameters, or create new ones, the environment is unaffected, unless you use the **export** command to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

You may augment the environment for any simple command by prefixing it with one or more assignments to parameters. Thus, the following two lines are equivalent:

```
TERM=450 cmd args
(export TERM; TERM=450; cmd args)
```

SIGNALS

sh ignores the INTERRUPT and QUIT signals for an invoked command if the command is followed by &. Otherwise, signals have the values inherited by the shell from its parent (see also **trap**).

EXECUTION

Each time a command is executed, the above substitutions are carried out. Except for the special commands listed below, a new process is created, and an attempt is made to execute the command via an **execve**(2).

The $PATH shell parameter defines the search path for the directory containing the command. Each alternative directory name is separated by a colon (:). The default path is **/usr/ucb:/bin:/usr/bin:/usr/apollo/bin**. If the command name contains a /, **sh** does not use the search path. Otherwise, **sh** searches each directory in the path for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell (i.e., a separate process) is spawned to read it. A command in parentheses is also executed in a sub-shell.

SPECIAL COMMANDS

The following commands are executed in the shell process, and except where specified, no input/output redirection is permitted for such commands.

#	For non-interactive shells, treat everything following the # as a comment, i.e., ignore the rest of the line. For interactive shells, the # has no special effect.
:	No effect; the command does nothing.

. *file* Read and execute commands from *file* and return. Use the search path $PATH to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified, break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified, resume at the *n*th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The $HOME shell parameter is the default *arg*.

eval [*arg* ...]
Read the arguments as input to the shell and execute the resulting command(s).

exec [*arg* ...]
Execute the command specified by the arguments in place of this shell without creating a new process. You may supply input/output arguments, and if you give no other type of arguments, **sh** modifies the input/output.

exit [*n*] If the shell is not interactive, exit with the exit status specified by *n*. If you omit *n*, the exit status is that of the last command executed. (An end-of-file will also exit from the shell.)

export [*name* ...]
Mark the given names for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, print a list of exportable names.

login [*arg* ...]
Equivalent to an **exec login** *arg* ... command.

read *name* ...
Read one line from the standard input. Assign successive words of the input to the variables *name* in order, with leftover words to the last variable. The return code is 0 unless the end-of-file is encountered.

readonly [*name* ...]
Mark the given names as read-only. You cannot change the values of these names by subsequent assignment. If no arguments are given, print a list of all read-only names.

rootnode [*arg*]
Change the current node entry directory to *arg*.

set [**–ekntuvx** [*arg* ...]]
–e Exit immediately if a command fails (on some systems, this switch works only on shells that are not interactive).

−k Place all keyword arguments in the environment for a command, not just those that precede the command name. The following, used in the shell, prints *a=b c* and *c*:

```
echo a=b c
set −k
echo a=b c
```

−n Read commands but do not execute them.

−t Exit after reading and executing one command.

−u Treat unset variables as an error when substituting.

−v Print shell input lines as they are read.

−x Print commands and their arguments as they are executed.

− Turn off the −x and −v options.

You can also use these options when you first invoke the shell. The current set may be found in $− .

Remaining arguments are positional parameters. The shell assigns them, in order, to **$1**, **$2**, etc. If no arguments are given, it prints the values of all names.

shift The positional parameters from **$2**... are renamed **$1**...

times Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
Read and execute *arg* upon receipt of signal(s) *n*. (Scan *arg* once when the trap is set and once when the trap is taken.) Execute **trap** commands in order of signal number. If *arg* is absent, reset all trap(s) *n* to their original values. If *arg* is the null string, the shell and invoked commands ignore this signal. If *n* is 0, execute the command *arg* on exit from the shell. Otherwise, execute the command upon receipt of signal *n* as numbered in **sigvec**(2). With no arguments, **trap** prints a list of commands associated with each signal number.

umask [*nnn*]
Set the user file creation mask to the octal value *nnn*. See **umask**(2) for details. If *nnn* is omitted, print the current value of the mask.

ver [*systype*[*command*]]
With no arguments, return the current value of the SYSTYPE environment variable that specifies the version of UNIX commands being executed by the shell. With a *systype* argument, change the SYSTYPE environment variable to either **bsd4.3** or **sys5.3**, depending on which is specified.

wait [*n*]

Wait for the specified process and report its termination status. If *n* is not given, wait for all currently active child processes. The return code from this command is that of the process being awaited.

inlib *pathname*

Install a user-supplied library specified by *pathname* in the current (shell) process. The library is used to resolve external references of programs (and libraries) loaded after its installation. Note that the library is not loaded into the address space unless it is needed to resolve an external reference. The list of inlibed libraries is passed to all children of the current shell. Use **llib**(1) to examine this list.

COMMAND LINE OPTIONS

If the first character of argument zero is –, **sh** reads commands from $HOME/**.profile** , if such a file exists. It then reads commands as described below. The following options are interpreted by the shell when it is invoked.

–c *string* Read commands from *string*.

–s Read commands from the standard input. Write shell output to file descriptor 2. (Note that the same activity occurs if no arguments remain.)

–i Make the shell *interactive*. (Note that this also occurs if the shell input and output are attached to a terminal, as told by *gtty*.) Ignore the terminate signal SIGTERM, so that **kill 0** does not kill the interactive shell. Catch and ignore the interrupt signal SIGINT, so that **wait** is interruptible. In all cases, ignore SIGQUIT.

–D*name=value*

Use the **–D** option to specify a parameter *name*, that will be set to *value*, then passed into the shell's environment. This Domain/OS SysV option is useful for tailoring the environment of a shell invoked from a program that is not another shell (such as the Display Manager). If you set the ENV parameter using this option, the startup script it specifies will be run. For example, if you define your <SHELL> key as follows: **cp /bin/sh –DENV=~/.shrc.pad**, the **~/.shrc.pad** script can contain commands to perform special processing for the pad and the shell. You can specify more than one **–D** option.

The remaining flags and arguments are described under the **set** command.

FILES

$HOME/**.profile**
/tmp/sh*
/dev/null

DIAGNOSTICS

Errors detected by the shell, such as those that occur in syntax, cause **sh** to return a nonzero exit status. If the shell is not being used interactively, then execution of the shell file is abandoned. Otherwise, the exit status of the last command executed is returned. Refer to **exit** under SPECIAL COMMANDS for more information.

BUGS

If << is used to provide standard input to an asynchronous process invoked by &, the shell gets confused about naming the input document. It creates a garbage file called **/tmp/sh*** and complains about not being able to find the file by another name.

SEE ALSO

csh(1), ksh(1), rootnode(1), test(1), execve(2), environ(7)

NAME

size – print section sizes in bytes of common object files

SYNOPSIS

size [**–n**] [**–f**] [**–o**] [**–x**] [**–V**] *files*

DESCRIPTION

size produces section size information in bytes for each loaded section in the common object files. The size of the text, data, and bss (uninitialized data) sections is printed, as well as the sum of the sizes of these sections. If an archive file is input to the size command the information for all archive members is displayed.

OPTIONS

–n	Includes NOLOAD sections in the size.
–f	Produces full output, that is, it prints the size of every loaded section, followed by the section name in parentheses.
–o	Prints numbers in octal, rather than decimal.
–x	Prints numbers in hexadecimal, rather than decimal.
–V	Supplies version information.

CAVEAT

Since the size of bss sections is not known until link-edit time, the *size* command will not give the true total size of pre-linked objects.

DIAGNOSTICS

size: name: cannot open
  if *name* cannot be read.

size: name: bad magic
  if *name* is not an appropriate common object file.

SEE ALSO

cc(1), ld(1), a.out(4), ar(4).

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

sleepfl suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

(**sleep 105**; *command*)&

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

SEE ALSO

alarm(2), sleep(3C) in the *SysV Programmer's Reference*.

NAME

sort – sort and/or merge files

SYNOPSIS

sort [**–cmu**] [–o*output*] [–y*kmem*] [–z*recsz*] [**–dfiMnr**] [**–btx**] [+*pos1* [–*pos2*]] [*files*]

DESCRIPTION

sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if – is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

OPTIONS

Options that alter default behavior:

–c Checks that the input file is sorted according to the ordering rules; gives no output unless the file is out of sort.

–m Merges only, the input files are already sorted.

–u Unique: suppresses all but one in each set of lines having equal keys.

–o*output* The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **–o** and *output.*

–y*kmem* The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, **sort** begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, **sort** will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, –y0 is guaranteed to start with minimum memory. By convention, –y (with no argument) starts with maximum memory.

–z*recsz* The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **–c** or **–m** options, a popular system default size will be used. Lines longer than the buffer size will cause **sort** to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

Options that override default ordering rules:

–d ''Dictionary'' order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

–f Folds lower case letters into upper case.

–i Ignores characters outside the ASCII range 040-0176 in non-numeric comparisons.

–M Compares as months. The first three non-blank characters of the field are folded to upper case and compared so that "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The **–M** option implies the **–b** option (see below).

–n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **–n** option implies the **–b** option (see below). Note that the **–b** option is only effective when restricted sort key specifications are in effect.

–r Reverses the sense of comparisons.

Options that alter the treatment of field separators:

–b Ignores leading blanks when determining the starting and ending positions of a restricted sort key. If the **–b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *–pos2* argument (see below).

–t*x* Uses *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (for example, *xx* delimits an empty field).

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation *+pos1 –pos2* restricts a sort key to one beginning at *pos1* and ending just before *pos2*. The characters at position *pos1* and just before *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *–pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field.

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfinr**. A starting position specified by *+m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m*+1st field; *+m.*0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by *–m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*+1st field; *–m.*1 **b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 –2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort –r –o outfile +1.0 –1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

```
sort –r +1.0b –1.1b infile1 infile2
```

Print the password file (*passwd*(4)) sorted by the numeric user ID (the third colon-separated field):

```
sort –t: +2n –3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **–um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort –um +2 –3 infile
```

WARNINGS

Comments and exits with non-zero status for various trouble conditions (for example, when input lines are too long), and for disorder discovered under the **–c** option. When the last line of an input file is missing a **new-line** character, **sort** appends one, prints a warning message, and continues.

sort does not guarantee preservation of relative line ordering on equal keys.

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

NAME

spell, hashmake, spellin, hashcheck – find spelling errors

SYNOPSIS

spell [–v] [–b] [–x] [–l] [*+local_file*] [*files*]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

spell ignores most **troff**(1), **tbl**(1), and **eqn**(1) constructions.

By default, **spell** follows chains of included files (**.so** and **.nx** **troff**(1) requests), *unless* the names of such included files begin with **/usr/lib**. Under the **–l** option, **spell** will follow the chains of *all* included files.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy–y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by **spell**:

hashmake	Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.
spellin	Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.
hashcheck	Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

OPTIONS

The following options apply to **spell**:

–v	Prints all words not literally in the spelling list, and indicate plausible derivations from the words in the spelling list.
–b	Checks British spelling. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

−x	Prints every plausible stem with = for each word.
+local_file	Removes words found in *local_file* are removed from **spell**'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to **spell**'s own spelling list) for each job.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]
Hashed spelling lists, American & British

S_SPELL=/usr/lib/spell/hstop
Hashed stop list

H_SPELL=/usr/lib/spell/spellhist
History file

/usr/lib/spell/spellprog
Program

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.

SEE ALSO

sed(1), sort(1), tee(1).

NAME

spell, hashmake, spellin, hashcheck – find spelling errors

SYNOPSIS

spell [–v] [**–b**] [–x] [**–l**] [*+local_file*] [*files*]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

spell ignores most **troff**(1), **tbl**(1), and **eqn**(1) constructions.

By default, **spell** follows chains of included files (**.so** and **.nx troff**(1) requests), *unless* the names of such included files begin with **/usr/lib**. Under the **–l** option, **spell** will follow the chains of *all* included files.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy–y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by **spell**:

hashmake	Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.
spellin	Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.
hashcheck	Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

OPTIONS

The following options apply to **spell**:

–v	Prints all words not literally in the spelling list, and indicate plausible derivations from the words in the spelling list.
–b	Checks British spelling. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

−x	Prints every plausible stem with = for each word.
+local_file	Removes words found in *local_file* are removed from spell's output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to spell's own spelling list) for each job.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]
Hashed spelling lists, American & British

S_SPELL=/usr/lib/spell/hstop
Hashed stop list

H_SPELL=/usr/lib/spell/spellhist
History file

/usr/lib/spell/spellprog
Program

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.

SEE ALSO

sed(1), sort(1), tee(1).

NAME

spline – interpolate smooth curve

SYNOPSIS

spline [options]

DESCRIPTION

spline takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by **graph**(1G).

The following *options* are recognized, each as a separate argument:

–**a** Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

–**k** The constant *k* used in the boundary value computation:

$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$

is set by the next argument (default $k = 0$).

–**n** Space output points so that approximately *n* intervals occur between the lower and upper *x* limits (default $n = 100$).

–**p** Make output periodic, i.e., match derivatives at ends. First and last input values should normally agree.

–**x** Next 1 (or 2) arguments are lower (and upper) *x* limits. Normally, these limits are calculated from the data. Automatic abscissas start at lower limit (default 0).

DIAGNOSTICS

When data is not strictly monotone in *x*, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1,000 input points is enforced silently.

SEE ALSO

graph(1G).

NAME

split – split a file into pieces

SYNOPSIS

split [–*n*] [file [name]]

DESCRIPTION

split reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically, up to **zz** (a maximum of 676 files). *Name* cannot be longer than 12 characters. If no output name is given, **x** is default.

If no input file is given, or if – is given in its stead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

NAME

start_sh – start a log-in shell

SYNOPSIS

start_sh
start_csh
start_ksh
start_rsh

DESCRIPTION

When you log in, the first shell is a log–in shell, therefore, the first argument is preceded by a –. This first shell runs your **.profile** or **.login file**. There should only be a single "log-in shell" created when you first log in. Other shells could be created with the DM commands like:

cp /bin/ksh - DENV=~/.kshrc.pad cp /bin/sh - DENV=~/.shrc.pad cp /bin/csh - DNEWPAD=true

See csh(1) for more information about the C shell.

NOTE

start_csh starts a log-in C shell (one that reads your **.login** file). **start_csh** is supported for compatibility with previous releases, and is not the recommended command to use at SR10.

SEE ALSO

csh(1), sh(1), ksh(1)

NAME

stat – statistical network useful with graphical commands

SYNOPSIS

node-name [*options*] [*files*]

DESCRIPTION

stat is a collection of command level functions (nodes) that can be interconnected using **sh(1)** to form a statistical network. The nodes reside in **/usr/bin/graf** (see **graphics(1G)**). Data is passed through the network as sequences of numbers (vectors), where a number is of the form:

[sign](digits)(.digits)[e[sign]digits]

evaluated in the usual way. Brackets and parentheses surround fields. All fields are optional, but at least one of the fields surrounded by parentheses must be present. Any character input to a node that is not part of a number is taken as a delimiter.

stat nodes are divided into four classes.

Transformers	Map input vector elements into output vector elements
Summarizers	Calculate statistics of a vector
Translators	Convert among formats
Generators	Sources of definable vectors.

Below is a list of synopses for **stat** nodes. Most nodes accept options indicated by a leading minus (–). In general, an option is specified by a character followed by a value, such as **c5**. This is interpreted as **c** := 5 (**c** is assigned 5). The following keys are used to designate the expected type of the value:

c	Characters
i	Integer
f	Floating point or integer
file	File name
string	String of characters, surrounded by quotes to include a *shell* argument delimiter.

Options without keys are flags. All nodes except *generators* accept files as input, hence it is not indicated in the synopses.

Transformers:

abs [–c*i*] – absolute value
columns (similarly for –c options that follow)

af [–c*i* t v] – arithmetic function
titled output, verbose

ceil	[–c*i*] – round up to next integer
cusum	[–c*i*] – cumulative sum
exp	[–c*i*] – exponential
floor	[–c*i*] – round down to next integer
gamma	[–c*i*] – gamma
list	[–c*i* d*string*] – list vector elements delimiter(s)
log	[–c*i* b*f*] – logarithm base
mod	[–c*i* m*f*] – modulus modulus
pair	[–c*i* F*file* x*i*] – pair elements File containing base vector, x group size
power	[–c*i* p*f*] – raise to a power power
root	[–c*i* r*f*] – take a root root
round	[–c*i* p*i* s*i*] – round to nearest integer, .5 rounds to 1 places after decimal point, significant digits
siline	[–c*i* i*f* n*i* s*f*] – generate a line given slope and intercept intercept, number of positive integers, slope
sin	[–c*i*] – sine
subset	[–a*f* b*f* c*i* F*file* i*i* l*f* nl np p*f* s*i* t*i*] – generate a subset above, below, File with master vector, interval, leave, master contains element numbers to leave, master contains element numbers to pick, pick, start, terminate

Summarizers:

bucket	[–a*i* c*i* F*file* h*f* i*i* l*f* n*i*] – break into buckets average size, File containing bucket boundaries, high, interval, low, number Input data should be sorted
cor	[–F*file*] – correlation coefficient File containing base vector
hilo	[– h l o ox oy]– find high and low values high only, low only, option form, option form with x prepended, option form with y prepended

lreg [–F*file* i o s] – linear regression
File containing base vector, intercept only, option form for *siline*, slope only

mean [–f*f* n*i* p*f*] – (trimmed) arithmetic mean
fraction, number, percent

point [–f*f* n*i* p*f* s] – point from empirical cumulative density function
fraction, number, percent, sorted input

prod – internal product

qsort [–c*i*] – quick sort

rank – vector rank

total – sum total

var – variance

Translators:

bar [–a b f g r*i* w*i* x*f* xa y*f* ya yl*f* yh*f*] – build a bar chart
suppress axes, bold, suppress frame, suppress grid, region, width in percent, x origin, suppress x-axis label, y origin, suppress y-axis label, y-axis lower bound, y-axis high bound
Data is rounded off to integers.

hist [–a b f g r*i* x*f* xa y*f* ya yl*f* yh*f*] – build a histogram
suppress axes, bold, suppress frame, suppress grid, region, x origin, suppress x-axis label, y origin, suppress y-axis label, y-axis lower bound, y-axis high bound

label [–b c F*file* h p r*i* x xu y yr] – label the axis of a GPS file
bar chart input, retain case, label File, histogram input, plot input, rotation, x-axis, upper x-axis, y-axis, right y-axis

pie [–b o p pn*i* pp*i* r*i* v x*i* y*i*] – build a pie chart
bold, values outside pie, value as percentage(:=100), value as percentage(:=i), draw percent of pie, region, no values, x origin, y origin
Unlike other nodes, input is lines of the form

> [< i e f c*c* >] value [label]
> ignore (do not draw) slice, explode slice, fill slice, color slice *c*=(black, red, green, blue)

plot [–a b c*string* d f F*file* g m r*i* x*f* xa xi*f* xh*f* xl*f* xn*i* xt y*f* ya yi*f* yh*f* yl*f* yn*i* yt] – plot a graph
suppress axes, bold, plotting characters, disconnected, suppress frame, File containing x vector, suppress grid, mark points, region, x origin, suppress x-axis label, x interval, x high bound, x low bound, number of ticks on x-axis, suppress x-axis title, y origin, suppress y-

axis label, y interval, y high bound, y low bound, number of ticks on y-axis, suppress y-axis title

title	[−b c l*string* v*string* u*string*] – title a vector or a GPS title bold, retain case, lower title, upper title, vector title

Generators:

gas	[−c*i* i*f* n*i* s*f* t*f*] – generate additive sequence interval, number, start, terminate
prime	[-c*i* h*i* l*i* n*i*] – generate prime numbers high, low, number
rand	[−c*i* h*f* l*f* m*f* n*i* s*i*] – generate random sequence high, low, multiplier, number, seed

RESTRICTIONS

Some nodes have a limit on the size of the input vector.

SEE ALSO

graphics(1G).
gps(4) in the *SysV Programmer's Reference*.

NAME

stcode – translate status code value to text message

SYNOPSIS

stcode *hex_stat_code*

DESCRIPTION

stcode prints the text message associated with a hexadecimal status code. This command is useful when a user program produces a hexadecimal status code instead of the textual message.

stcode processes predefined status codes. No provision is currently made to add user-defined status codes to the error text database.

hex_stat_code (required)	Specifies hexadecimal status code to be translated.

EXAMPLES

```
$ stcode 80001
disk not ready (from OS / disk manager)
```

NAME

strinfo – prints STREAMS-related information

SYNOPSIS

strinfo [**–v**] [**–s**] [**–d** *attributes*] [**–m** *attributes*]
[**–q** *attributes*] [**–t** *attributes*]

DESCRIPTION

strinfo prints information about configured STREAMS modules and drivers, currently active queues and Streams, and cumulative usage statistics. **getopt**(2) standards are not followed; flags must appear separately in order to be recognized.

OPTIONS

–v Verbose output.

–s Print cumulative usage statistics.

–d *attributes* Print information about configured drivers. *attributes* is a blank-separated list of one or more of the following:

all	Print all attributes (default).
index	Index in the **cdevsw** or **fmodsw** table.
majdev	Major device number.
id	Id number.
name	Name.
stats	Statistics.
privstats	Private statistics.
type	Object type of device.
traits	Traits supported.
psz	Minimum and maximum packet size.
water	Low and high water marks.
write	Print write-side information.

–m *attributes* Print information about configured modules. *attributes* is a blank separated list of one or more of the attributes listed above for **–d**.

–q *attributes* Print information about active queues. *attributes* is a blank separated list of one or more of the following:

all	Print all attributes (default).
address	Address of the queue.
name	Name of the module.
next	Address of next queue.

count	Count of message blocks on queue.
flags	Queue state.
psz	Minimum and maximum packet size.
water	Low and high water mark.
msgs	Print contents of messages on queue.

−t *attributes* Print information about active Streams. *attributes* is a blank-separated list of one or more of the following:

all	Print all attributes (default).
index	Index into the Stream table.
address	Address of the Stream.
wrq	Address of Stream head read and write queues.
iocblk	Return block for **ioctl**.
inode	Inode pointer.
strtab	Pointer to the **streamtab** structure for the Stream.
flag	State/flags.
iocid	**ioctl** id.
iocwait	Count of processes waiting to do **ioctl**.
pgrp	Process group, for signals.
wroff	Write offset.
error	Hangup or error to set **u.u_error**.
pushcnt	Number of pushes done on Stream.
list	Pointer to list of processes to receive SIGPOLL and to wake up **poll**.
sigflags	Logical OR of all signal list events.
pollflags	Logical OR of all poll list events.

SEE ALSO

trmon(1), trconf(1)

NAME

strip – strip symbol and line number information from a common object file

SYNOPSIS

strip [–l] [–x] [–b [–r] [–V] [–Aa] *filename* . . .

DESCRIPTION

The **strip** command strips the symbol table and line number information from common object files, including archives. Once this has been done, no symbolic debugging access is available for that file; therefore, this command is normally run only on production modules that have been debugged and tested. By default, enough information is saved to allow traceback information to be obtained from stripped files.

The amount of information stripped from the symbol table can be controlled by using any of the *options* listed below:

OPTIONS

–l	Strips line number information only; does not strip any symbol table information.
–x	Does not strip static or external symbol information.
–b	Same as the –x *option*, but also does not strip scoping information (e.g., beginning and end of block delimiters).
–r	Does not strip static or external symbol information, or relocation information.
–V	Prints the version of the **strip** command executing on the standard erroı output.
–Aa	Strips .blocks and .lines sections of common object files so that traceback information is no longer available.

If there are any relocation entries in the object file and any symbol table information iѕ to be stripped, **strip** complains and terminates without stripping *filename* unless the –ı *option* is used.

If you use **strip** on a common archive file [see **ar**(4)] the archive symbol table anc module table are removed. You must restore the archive symbol table by executinɡ **ar**(1) with the s *option* before the archive can be link-edited by **ld**(1). **strip** produce: appropriate warning messages when this situation arises.

strip is used to reduce the file storage overhead taken by the object file.

FILES

TMPDIR/strp*	Temporary files

TMPDIR is usually **/usr/tmp** but can be redefined by setting the environment variabl **TMPDIR** [see **tempnam()** in **tmpnam(3S)**].

DIAGNOSTICS

strip: name: cannot open
If *name* cannot be read.

strip: name: bad magic
If *name* is not an appropriate common object file.

SEE ALSO

ar(1), cc(1), ld(1), ts(1), tmpnam(3S), a.out(4), ar(4).

NAME

stty – set the options for a terminal

SYNOPSIS

stty [**–a**] [**–g**] [*options*]

DESCRIPTION

stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

In this report, if a character is preceded by a caret (^), then the value of that option is the corresponding CTRL character (e.g., "^H" is **CTRL/H** ; in this case, recall that **CTRL/H** is the same as the "back-space" key.) The sequence "^´" means that an option has a null value. This has no effect on Apollo transcript pads. It is useful on dialup terminals or VT100 windows. **stty –a**

–a	Reports all of the option settings;
–g	Reports current settings in a form that can be used as an argument to another **stty** command.

Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (–parenb)	Enable (disable) parity generation and detection.
parodd (–parodd)	Select odd (even) parity.
cs5 cs6 cs7 cs8	Select character size (see **termio**(7)).
0	Hang up phone line immediately.
110 300 600 1200 1800 2400 4800 9600 19200 38400	Set terminal baud rate to the number given, if possible. (Al speeds are not supported by all hardware interfaces.)
hupcl (–hupcl)	Hang up (do not hang up) Dataphone connection on last close.
hup (–hup)	Same as **hupcl (–hupcl)**.
cstopb (–cstopb)	Use two (one) stop bits per character.
cread (–cread)	Enable (disable) the receiver.
clocal (–clocal)	n Assume a line without (with) modem control.
loblk (–loblk)	Block (do not block) output from a non-current layer.

Input Modes

ignbrk (–ignbrk)	Ignore (do not ignore) break on input.
brkint (–brkint)	Signal (do not signal) INTR on break.
ignpar (–ignpar)	Ignore (do not ignore) parity errors.
parmrk (–parmrk)	Mark (do not mark) parity errors (see **termio**(7)).
inpck (–inpck)	Enable (disable) input parity checking.

istrip (–istrip)	Strip (do not strip) input characters to seven bits.
inlcr (–inlcr)	Map (do not map) NL to CR on input.
igncr (–igncr)	Ignore (do not ignore) CR on input.
icrnl (–icrnl)	Map (do not map) CR to NL on input.
iuclc (–iuclc)	Map (do not map) upper-case alphabetics to lower case on input.
ixon (–ixon)	Enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.
ixany (–ixany)	Allow any character (only DC1) to restart output.
ixoff (–ixoff)	Request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (–opost)	Post-process output (do not post-process output; ignore all other output modes).
olcuc (–olcuc)	Map (do not map) lower-case alphabetics to upper case on output.
onlcr (–onlcr)	Map (do not map) NL to CR-NL on output.
ocrnl (–ocrnl)	Map (do not map) CR to NL on output.
onocr (–onocr)	Do not (do) output CRs at column zero.
onlret (–onlret)	On the terminal NL performs (does not perform) the CR function.
ofill (–ofill)	Use fill characters (use timing) for delays.
ofdel (–ofdel)	Fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	Select style of delay for carriage returns (see **termio**(7)).
nl0 nl1	Select style of delay for line-feeds (see **termio**(7)).
tab0 tab1 tab2 tab3	Select style of delay for horizontal tabs (see **termio**(7)).
bs0 bs1	Select style of delay for backspaces (see **termio**(7)).
ff0 ff1	Select style of delay for form-feeds (see **termio**(7)).
vt0 vt1	Select style of delay for vertical tabs (see **termio**(7)).

Local Modes

isig (–isig)	Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.
icanon (–icanon)	Enable (disable) canonical input (ERASE and KILL processing).
xcase (–xcase)	Canonical (unprocessed) upper/lower-case presentation.
echo (–echo)	Echo back (do not echo back) every character typed.
echoe (–echoe)	Echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* Keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

echok (−echok)	Echo (do not echo) NL after KILL character.
lfkc (−lfkc)	The same as **echok (−echok)**; Obsolete.
echonl (−echonl)	Echo (do not echo) NL.
noflsh (−noflsh)	Disable (enable) flush after INTR, QUIT, or SWTCH.
stwrap (−stwrap)	Disable (enable) truncation of lines longer than 79 characters on a synchronous line.
stflush (−stflush)	Enable (disable) flush on a synchronous line after every **write**(2).
stappl (−stappl)	Use application mode (use line mode) on a synchronous line.

Control Assignments

control-character c	Set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **swtch**, **eof**, **ctab**, **min**, or **time** (**ctab** is used with **−stappl**; **min** and **time** are used with **−icanon**; see **termio**(7)). If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., "^**D**" is a **CTRL/D**); "^?" is interpreted as DEL and "^−" is interpreted as undefined.
line *i*	Set line discipline to *i* (0 < *i* < 127).

Combination Modes

evenp or **parity**	Enable **parenb** and **cs7**.
oddp	Enable **parenb**, **cs7**, and **parodd**.
−parity, **−evenp**, or **−oddp**	Disable **parenb**, and set **cs8**.
raw (**−raw** or **cooked**)	Enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).
nl (−nl)	Unset (set) **icrnl**, **onlcr**. In addition **−nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (−lcase)	Set (unset) **xcase**, **iuclc**, and **olcuc**.
LCASE (−LCASE)	Same as **lcase (−lcase)**.
tabs (**−tabs** or **tab3**)	Preserve (expand to spaces) tabs when printing.
ek	Reset ERASE and KILL characters back to normal # and @.
sane	resets all modes to some reasonable values.
term	Set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

SEE ALSO

tabs(1).

ioctl(2) in the *SysV Programmer's Reference*.

termio(7) in the *Managing SysV System Software*.

NAME

su – become super-user or another user

SYNOPSIS

su [–] [*name* [*arg* ...]]

DESCRIPTION

su allows you to be recognized as another user without logging off. The default user *name* is **root** (i.e., super-user).

su requires that you supply the appropriate password (unless you are already **root**). If the password is correct, **su** executes a new shell with the real and effective user ID set to that of the specified user. The new shell is the optional program named in the shell field of the specified user's password file entry, or **/bin/sh** if none is specified. See **passwd**(4) and **sh**(1) for more information. To restore normal user ID privileges, type an EOF (CTRL/D) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like **sh**(1), an argument of the form **–c** *string* executes *string* via the shell, and an argument such as **–r** gives you a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like **sh**(1). If the first argument to **su** is a dash (–), the environment is changed to what would be expected if you actually logged in as the specified user. This is done by invoking the program used as the shell with an argument value whose first character is –, thus causing first the system's profile (/etc/**profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of **$PATH**, which is set to **/bin:/etc:/usr/bin:/usr/apollo/bin** for **root**.

Note that if the optional program used as the shell is **/bin/sh**, your **.profile** can check *arg0* for **–sh** or **–su** to determine if it was invoked by **login**(1)
or **su** respectively. If your program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *–program* by both **login**(1) and **su**.

All attempts to become another user using **su** are logged in the log file /usr/**adm**/sulog.

EXAMPLES

To become user *bin* while retaining your previously exported environment, execute the following:

```
su bin
```

To become user *bin* but change the environment to what would be expected if *bin* had originally logged in, execute:

```
su – bin
```

To execute *command* with the temporary environment and permissions of user *bin*, type:

```
su – bin –c command args
```

FILES

/etc/passwd	System's password file
/etc/profile	System's profile
$HOME.profile	User's profile
/usr/adm/sulog	Log file

SEE ALSO

env(1), **login**(1), **sh**(1), **passwd**(4), **environ**(5).

NAME

sum – print checksum and block count of a file

SYNOPSIS

s um [**–r**] *file*

DESCRIPTION

sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **–r** causes an alternate algorithm to be used in computing the checksum.

DIAGNOSTICS

"*Read error*" is indistinguishable from end of file on most devices; check the block count.

SEE ALSO

wc(1).

NAME

swapul – rearrange underlining

SYNOPSIS

swapul

DESCRIPTION

swapul reads lines from standard input, rearranges underlining so that underlines follow a character in the output stream (instead of being preceded by them), and writes the resulting text to standard output.

NAME

swedish_to_iso – convert files to ISO format

SYNOPSIS

swedish_to_iso *input_file output_file*

DESCRIPTION

These utilities convert files written with the overloaded 7-bit national fonts to the Internation Standards Organization (ISO) 8-bit format. The overloaded fonts include any with a specific language suffix (for example, **f7x13.french**, or **din_f7x11.german**). If you created and/or edited a file using one of the national fonts, that file is a candidate for conversion.

You are not required to convert files, but probably will want to because

1. The overloaded fonts replace certain ASCII characters with national ones, have that subset of ASCII characters and the national characters in one file. The 8-bit fonts available as of SR10 include all the ASCII characters and the national characters.
2. The 8-bit fonts also include a wider range of national characters, so you can enter any character in any western European language. This was not always possible with the overloaded fonts. For example, there was not enough space in the overloaded font to include all the French characters, but they all exist in the 8-bit fonts.

There are two parameters to the conversion utilities. You must provide the name of the file you want to convert (*input_file*) and your *output_file*. If *output_file* already exists, the utilities abort.

The default location for the utilities is **/usr/apollo/bin**.

FILES

/usr/apollo/bin/french_to_iso	Converts overloaded French to ISO format
/usr/apollo/bin/german_to_iso	Converts overloaded German to ISO format
/usr/apollo/bin/nor.dan_to_iso	Converts overloaded Norwegian/Danish to ISO format
/usr/apollo/bin/swedish_to_iso	Converts overloaded Swedish/Finnish to ISO format
/usr/apollo/bin/swiss_to_iso	Converts overloaded Swiss to ISO format
/usr/apollo/bin/uk_to_iso	Converts overloaded U.K. English to ISO format

DIAGNOSTICS

All messages are generally self-explanatory.

NAME

swiss_to_iso – convert files to ISO format

SYNOPSIS

swiss_to_iso *input_file output_file*

DESCRIPTION

These utilities convert files written with the overloaded 7-bit national fonts to the Internation Standards Organization (ISO) 8-bit format. The overloaded fonts include any with a specific language suffix (for example, **f7x13.french**, or **din_f7x11.german**). If you created and/or edited a file using one of the national fonts, that file is a candidate for conversion.

You are not required to convert files, but probably will want to because

1. The overloaded fonts replace certain ASCII characters with national ones, have that subset of ASCII characters and the national characters in one file. The 8-bit fonts available as of SR10 include all the ASCII characters and the national characters.
2. The 8-bit fonts also include a wider range of national characters, so you can enter any character in any western European language. This was not always possible with the overloaded fonts. For example, there was not enough space in the overloaded font to include all the French characters, but they all exist in the 8-bit fonts.

There are two parameters to the conversion utilities. You must provide the name of the file you want to convert (*input_file*) and your *output_file*. If *output_file* already exists, the utilities abort.

The default location for the utilities is **/usr/apollo/bin**.

FILES

/usr/apollo/bin/french_to_iso	Converts overloaded French to ISO format
/usr/apollo/bin/german_to_iso	Converts overloaded German to ISO format
/usr/apollo/bin/nor.dan_to_iso	Converts overloaded Norwegian/Danish to ISO format
/usr/apollo/bin/swedish_to_iso	Converts overloaded Swedish/Finnish to ISO format
/usr/apollo/bin/swiss_to_iso	Converts overloaded Swiss to ISO format
/usr/apollo/bin/uk_to_iso	Converts overloaded U.K. English to ISO format

DIAGNOSTICS

All messages are generally self-explanatory.

NAME

sync – forces write to disk

SYNOPSIS

sync

DESCRIPTION

The **sync** command executes the **sync** system primitive. It flushes all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point are saved.

The **sync** operation is not actually necessary on DOMAIN hardware, because the system buffers are automatically written to disk at shutdown. Nevertheless, we provide it in the interest of ensuring compatibility.

SEE ALSO

sync(2)

NAME

systype – display version stamp

SYNOPSIS

/usr/apollo/bin/systype *file*

DESCRIPTION

systype displays the UNIX version stamp of the specified object file. Four columns are displayed in the output. The first (OBJTYPE) contains the object type, the second (SYSTYPE) contains the **systype** (one of the following):

none
any
sys5
sys5.3
bsd4.2
bsd4.3

The third column (RUNTYPE) contains the runtype, which has the same set of possible types as the **systype**; and the forth column (FILE) is the name of the file.

The format is as follows:

OBJTYPE	SYSTYPE	RUNTYPE	FILE
coff	bsd4.3	sys5.3	/bin/cc
obj	bsd4.2		//node_1/bsd4.2/bin/cc

Note that files of object type OBJ, do not have runtypes.

EXAMPLES

$ systype /bin/cc

```
OBJTYPE         SYSTYPE         RUNTYPE     FILE
coff            bsd4.3          sys5.3      /bin/cc
```

$ systype /bsd4.2/bin/cc

```
OBJTYPE         SYSTYPE         RUNTYPE     FILE
obj             bsd4.2                      /bsd4.2/bin/cc
```

SEE ALSO

cc(1), ld(1);
Using Your SysV Environment

NAME

tabs – set tabs on a terminal

SYNOPSIS

tabs [tabspec] [–Ttype] [+mn]

DESCRIPTION

tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs. This has no effect on Apollo transcript pads. It is useful on connected terminals and VT100 windows.

tabspec Four types of tab specification are accepted for *tabspec*. They are described below: canned (*–code*), repetitive (*–n*), arbitrary (*n1,n2,...*), and file (*––file*). If no **tabspec** is given, the default value is **–8**, i.e., UNIX system "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

–code Use one of the codes listed below to select a *canned* set of tabs. The legal codes and their meanings are as follows:

–a 1,10,16,36,72
Assembler, IBM S/370, first format

–a2 1,10,16,40,72
Assembler, IBM S/370, second format

–c 1,8,12,16,20,55
COBOL, normal format

–c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows (see fspec(4)):

<:t–c2 m6 s66 d:>

–c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than **–c2**. This is the recommended format for COBOL. The appropriate format specification is (see fspec(4)):

<:t–c3 m6 s66 d:>

–f 1,7,11,15,19,23
FORTRAN

–p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

–s 1,10,55
SNOBOL

–u 1,12,20,44
UNIVAC 1100 Assembler

–n A *repetitive* specification requests tabs at columns 1+*n*, 1+2**n*, etc. Of particular importance is the value **8**: this represents the UNIX system "standard" tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value **0**, implying no tabs at all.

n1,n2,... The *arbitrary* format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats **1,10,20,30**, and **1,10,+10,+10** are considered identical.

—file If the name of a *file* is given, **tabs** reads the first line of the file, searching for a format specification (see **fspec**(4)). If it finds one there, it sets the tab stops according to it, otherwise it sets them as **–8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the **pr**(1) command:

```
tabs — file; pr file
```

Any of the following also may be used; if a given flag occurs more than once, the last value given takes effect:

–T*type* **tabs** usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in **term**(5). If no **–T** flag is supplied, **tabs** uses the value of the environment variable **TERM**. If **TERM** is not defined in the *environment* (see **environ**(5)), **tabs** tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is **10**. For a TermiNet, the first value in the tab list should be **1**, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

EXAMPLES

tabs –a Example using *–code* (*canned* specification) to set tabs to the settings required by the IBM assembler: columns 1, 10, 16, 36, 72.

tabs –8 Example of using *–n* (*repetitive* specification), where *n* is **8**, Causes tabs to be set every eighth position:
1+(1*8), 1+(2*8), ... which evaluate to columns 9, 17, ...

tabs 1,8,36 Example of using *n1,n2,...* (*arbitrary* specification) to set tabs at columns 1, 8, and 36.

tabs —$HOME/fspec.list/att4425
Example of using *—file* (*file* specification) to indicate that tabs should be set according to the first line of *$HOME/fspec.list/att4425* (see fspec(4)).

NOTE

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin. **tabs** clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

WARNING

The **tabspec** used with the **tabs** command is different from the one used with the **newform**(1) command. For example, **tabs –8** sets every eighth position; whereas **newform –i–8** indicates that tabs are set every eighth position.

DIAGNOSTICS

illegal tabs	When arbitrary tabs are ordered incorrectly.
illegal increment	When a zero or missing increment is found in an arbitrary specification.
unknown tab code	When a *canned* code cannot be found.
can't open	If *—file* option used, and file can't be opened.
file indirection	If *—file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted.

SEE ALSO

newform(1), pr(1), tput(1).
fspec(4), terminfo(4), environ(5), term(5) in the *SysV Programmer's Reference*.

NAME

tail – deliver the last part of a file

SYNOPSIS

tail [±[number][**lbc**[**f**]]] [*file*]

DESCRIPTION

tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *−number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the −f (''follow'') option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail −f fred
```

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time **tail** is initiated and killed. As another example, the command:

```
tail −15cf fred
```

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time **tail** is initiated and killed.

BUGS

Tails relative to the end of the file are stored in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

WARNING

The **tail** command will only tail the last 4096 bytes of a file regardless of its line count.

SEE ALSO

dd(1M)

NAME

tar – tape file archiver

SYNOPSIS

/etc/**tar** –**c**[vwf**b**[*#s*]] *device block files* ...
/etc/**tar** –**r**[vwb[*#s*]] *device block* [*files* ...]
/etc/**tar** –**t**[vf[*#s*] *device*
/etc/**tar** –**u**[vwb[*#s*]] *device block* [*files* ...]
/etc/**tar** –**x**[lmovwf[*#s*]] *device* [*files* ...]

DESCRIPTION

tar saves and restores files on magnetic tape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing one function letter (**c**, **r**, **t**, **u**, or **x**) and possibly followed by one or more function modifiers (**v**, **w**, **f**, **b**, and **#**). Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r Replace. The named *files* are written on the end of the tape. The **c** function implies this function.

x Extract. The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. Use the file or directory's relative path when appropriate, or **tar** will not find a match. The owner, modification time, and mode are restored (if possible). If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.

t Table. The names and other information for the specified files are listed each time that they occur on the tape. The listing is similar to the format produced by the **ls –l** command. If no *files* argument is given, all the names on the tape are listed.

u Update. The named *files* are added to the tape if they are not already there, or have been modified since last written on that tape. This key implies the **r** key.

c Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This key implies the **r** key.

The characters below may be used in addition to the letter that selects the desired function. Use them in the order shown in the synopsis.

#s This modifier determines the drive on which the tape is mounted (replace # with the drive number) and the speed of the drive (replace s with **l**, **m**, or **h** for low, medium or high). The modifier tells **tar** to use a drive other than the default drive, or the drive specified with the -f option. For example, with the **5h** modifier, **tar** would use **/dev/mt/5h** or **/dev/mt0** instead of the default drives /dev/mt/0m or /dev/mt0, respectively. However, if for example, **–f /dev/rmt0 5h** appeared on the command line, **tar** would use **/dev/rmt5h** or

/devmt0. The default entry is **0m**.

v Verbose. Normally, **tar** does its work silently. The v (verbose) option causes it to type the name of each file it treats, preceded by the function letter. With the **t** function, v gives more information about the tape entries than just the name.

w What. This causes **tar** to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with y is given, the action is performed. Any other input means ''no''. This is not valid with the **t** key.

f File. This causes **tar** to use the *device* argument as the name of the archive instead of **/dev/mt/0m** or **/dev/mt0**. If the name of the file is –, **tar** writes to the standard output or reads from the standard input, whichever is appropriate. Thus, **tar** can be used as the head or tail of a pipeline. **tar** can also be used to move hierarchies with the command:

```
cd fromdir; tar cf – .
```

b Blocking Factor. This causes **tar** to use the *block* argument as the blocking factor for tape records. The default is 1, the maximum is 20. This function should not be supplied when operating on regular archives or block special devices. It is mandatory however, when reading archives on raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes created on block special devices (key letters x and **t**).

l Link. This tells **tar** to complain if it cannot resolve all of the links to the files being dumped. If **l** is not specified, no error messages are printed.

m Modify. This tells **tar** to not restore the modification times. The modification time of the file will be the time of extraction.

o Ownership. This causes extracted files to take on the user and group identifier of the user running the program, rather than those on tape. This is only valid with the x key.

A Include Apollo-specific information. Allows Domain/OS typed files.

BUGS

There is no way to ask for the *n*-th occurrence of a file.
Tape errors are handled ungracefully.
The **u** option can be slow.
The **b** option should not be used with archives that are going to be updated. The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the **b** option should not be used at all, because updating an archive stored on disk can destroy it.
The current limit on file name length is 100 characters.
tar doesn't copy empty directories or special files.

FILES

/dev/mt/*
/dev/mt*
/tmp/tar*
/dev/mt/ctape
/dev/mt/0m
/dev/rmt/0m

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

SEE ALSO

ar(1), cpio(1), ls(1), mt(1).

NAME

tb – print process traceback

SYNOPSIS

tb [*options*] [*process_spec*]

DESCRIPTION

tb prints a process traceback, listing the name and current line number of each routine on the call stack. There are two forms of traceback:

Active	Traces the current state of an executing process.
Diagnostic	Traces the state of an aborted process at the time of the fault which killed it.

process_spec (optional)

UNIX process ID (PID), aegis process name, or aegis process UID. Process names are not recorded in the process dump file, so dead processes must be referenced by PID or UID. Since PID's are reused multiple dump file entries for the same PID are possible, the command will select the most recent.

Default if omitted: perform a diagnostic traceback for the last child of the invoking process

OPTIONS

–p[roc]
Traces exactly the specified process. If this option is absent, the specified process or one of its children may be traced, as described below.

–d[iagnostic]
Prints a diagnostic traceback of an aborted process.

–n[ode] *node_spec*
Uses the process dump file on the specified node. Implies **–diagnostic.**

–c[ommand] *pathname*
Prints diagnostic traceback(s) for processes running the specified program. *pathname* must be reachable from the working directory; command search rules are not applied. Implies **–diagnostic.**

–s[ince] *date_time_spec*
Prints diagnostic traceback(s) for processes which aborted after the specified time. Implies **–diagnostic.** The format for *date_time_spec* is [[[yyyy/]mm/dd][.][hh:mm[:ss]].

–l[ast] [*n*]
Prints the *n* most recent entries in the process dump file (which also satisfy other selection criteria if given). *n* defaults to 1. If neither **–last** nor **–all** is specified **tb** prints only the most recent entry. Implies **–diagnostic.**

–a[ll]	Prints all entries in the process dump file (which also satisfy other selection criteria if given.) If neither **–last** or **–all** are specified, **tb** prints only the most recent entry. Implies **–diagnostic.**
–f[ull]	Prints additional fault diagnostic information, such as register values. Implies **–diagnostic.**
–b[rief]	Lists entries in the process dump file that satisfy selection criteria, but do not print tracebacks. The listing shows the process, parent, and group IDs, the time of the dump, the abort status, and the program that was running.
–t[asks]	Traces all tasks in the process. By default only the currently active task is shown. Ignored if tasking is not enabled. Applies only to active process tracebacks.
–h[eaders_off]	Suppresses output of process ID, dump time, and program name preceding diagnostic traceback, or of column headers in brief format. It has no effect on active process traceback.

Diagnostic Tracebacks

A diagnostic traceback shows the state of the call stack at the time of a fault which causes a process to be aborted. Traceback information is written to **'node_data/system_logs/proc_dump** at the time of the fault. This is a circular buffer in which the oldest information is overwritten as needed to make room for new. There is space for approximately 150-200 dumps. **tb** prints up to 128 call levels for diagnostic tracebacks.

tb prints a diagnostic traceback if the command line specifies **–diagnostic** or any option which implies it, or if the process specified is not active. If **–diagnostic** is specified together with an active process, the most recent aborted child of that process is traced (or most recent children if **–last** or **–all** is specified).

If no options are given (except possibly **–f**, **–b** or **–h**) **tb** prints a diagnostic traceback for the most recent aborted child of the process which invoked **tb**.

Examples of Requesting Diagnostic Tracebacks

Assume **process_5** is an active shell process, and process number 107 is not active. Traceback process 107.

```
$ tb 107
```

Traceback last aborted command invoked from **process_5**.

```
$tb –d process_5
```

Traceback last aborted command from this shell

$tb

Traceback last aborted process running test3

$tb –c test3

List all entries in the process dump file made today

$tb –s today –a –b

Active Process Tracebacks

An active process traceback shows the current state of an executing process, listing the name and line number of each procedure in the call stack. The process is suspended while the traceback is taken. **tb** prints an active process traceback if the command line specifies an active process and does not include **–diagnostic** (or any option that implies it). If the process is specified by name and has any active children, then the most recent child is traced. (This allows a process to be specified by the name of its invoking shell process.) This behavior may be overriden by the **–proc** switch, or by specifying the process by PID or UID. Note that the only other option applicable to active process tracebacks is **–task**.

Examples of Requesting Active Process Tracebacks

Assume **process_7** is an active shell process, from which a command running in process 747 has been invoked.

$tb 747

Traceback the invoked command

$tb process_7

same

$tb –p process_7

Traceback the shell process itself

NAME

gdev: **hpd**, **erase**, **hardcopy**, **tekset**, **td** – graphical device routines and filters

SYNOPSIS

hpd [– *options*] [GPS *file* ...]
erase
hardcopy
tekset
td [–**ern***n*] [GPS *file* ...]

DESCRIPTION

All of the commands described below reside in **/usr/bin/graf** (see **graphics**(1G)).

hpd Translate a GPS (graphical primitive string; see gps(4)) to instructions for the Hewlett-Packard 7221A Graphics Plotter. A viewing window is computed from the maximum and minimum points in *file* unless the **–u** or **–r** *option* is provided. If no *file* is given, the standard input is assumed.

hpd Options

c*n*	Select character set *n*, *n* between 0 and 5.
p*n*	Select pen numbered *n*, *n* between 1 and 4 inclusive.
r*n*	Window on GPS region *n*, *n* between 1 and 25 inclusive.
s*n*	Slant characters *n* degrees clockwise from the vertical.
u	Window on the entire GPS universe.
xd*n*	Set x displacement of the viewport's lower left corner to *n* inches.
xv*n*	Set width of viewport to *n* inches.
yd*n*	Set y displacement of the viewport's lower left corner to *n* inches.
yv*n*	Set height of viewport to *n* inches.

erase Send characters to a Tektronix 4010 series storage terminal to erase the screen.

hardcopy When issued at a Tektronix display terminal with a hard copy unit, **hardcopy** generates a screen copy on the unit.

tekset Send characters to a Tektronix terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.

td Translate a GPS to scope code for a Tektronix 4010 series storage terminal. A viewing window is computed from the maximum and minimum points in *file* unless the **−u** or **−r** *option* is provided. If no *file* is given, the standard input is assumed.

td Options

e	Do not erase screen before initiating display.
r*n*	Display GPS region *n*, *n* between 1 and 25 inclusive.
u	Display the entire GPS universe.

SEE ALSO

graphics(1G).
gps(4) in the *SysV Programmer's Reference*.

NAME

tee – pipe fitting

SYNOPSIS

tee [**–i**] [**–a**] [*file*] ...

DESCRIPTION

tee transcribes the standard input to the standard output and makes copies in the *files*.

OPTIONS

–i Ignore interrupts;

–a Causes the output to be appended to the *files* rather than overwriting them.

NAME

gdev: **hpd**, **erase**, **hardcopy**, **tekset**, **td** – graphical device routines and filters

SYNOPSIS

hpd [– *options*] [GPS *file* ...]
erase
hardcopy
tekset
td [–**ern***n*] [GPS *file* ...]

DESCRIPTION

All of the commands described below reside in **/usr/bin/graf** (see **graphics**(1G)).

hpd Translate a GPS (graphical primitive string; see **gps**(4)) to instructions for the Hewlett-Packard 7221A Graphics Plotter. A viewing window is computed from the maximum and minimum points in *file* unless the **–u** or **–r** *option* is provided. If no *file* is given, the standard input is assumed.

hpd Options

c*n*	Select character set *n*, *n* between 0 and 5.
p*n*	Select pen numbered *n*, *n* between 1 and 4 inclusive.
r*n*	Window on GPS region *n*, *n* between 1 and 25 inclusive.
s*n*	Slant characters *n* degrees clockwise from the vertical.
u	Window on the entire GPS universe.
xd*n*	Set x displacement of the viewport's lower left corner to *n* inches.
xv*n*	Set width of viewport to *n* inches.
yd*n*	Set y displacement of the viewport's lower left corner to *n* inches.
yv*n*	Set height of viewport to *n* inches.

erase Send characters to a Tektronix 4010 series storage terminal to erase the screen.

hardcopy When issued at a Tektronix display terminal with a hard copy unit, **hardcopy** generates a screen copy on the unit.

tekset Send characters to a Tektronix terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.

td Translate a GPS to scope code for a Tektronix 4010 series storage terminal. A viewing window is computed from the maximum and minimum points in *file* unless the −**u** or −**r** *option* is provided. If no *file* is given, the standard input is assumed.

td Options

e	Do not erase screen before initiating display.
r*n*	Display GPS region *n*, *n* between 1 and 25 inclusive.
u	Display the entire GPS universe.

SEE ALSO

graphics(1G).
gps(4) in the *SysV Programmer's Reference*.

NAME

telnet – user interface to the TELNET protocol

SYNOPSIS

telnet [*host* [*port*]]

DESCRIPTION

telnet is used to communicate with another host using the **TELNET** protocol. If **telnet** is invoked without arguments, it enters command mode, indicated by its prompt (''telnet>''). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command (see below) with those arguments.

Once a connection has been opened, **telnet** enters an input mode. The input mode entered will be either ''character at a time'' or ''line by line'' depending on what the remote system supports.

In ''character at a time'' mode, most text typed is immediately sent to the remote host for processing.

In ''line by line'' mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The ''local echo character'' (initially ''^E'') may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the **localchars** toggle is TRUE (the default in line mode; see below), the user's **quit**, **intr**, and **flush** characters are trapped locally, and sent as **TELNET** protocol sequences to the remote side. There are options (see **toggle autoflush** and **toggle autosynch** below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the **TELNET** sequence) and flush previous terminal input (in the case of **quit** and **intr**).

While connected to a remote host, **telnet** command mode may be entered by typing the **telnet** ''escape character'' (initially ''^]''). When in command mode, the normal terminal editing conventions are available.

COMMANDS

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the **mode**, **set**, **toggle**, and **display** commands).

open *host* [*port*]
: Open a connection to the named host. If no port number is specified, **telnet** will attempt to contact a **TELNET** server at the default port. The host specification may be either a host name (see **hosts**(4)) or an Internet address specified in ''dot notation'' (see **inet**(3N)).

close
: Close a **TELNET** session and return to command mode.

quit Close any open **TELNET** session and exit **telnet**. An end of file (in command mode) will also close a session and exit.

z Suspend **telnet**. This command only works when the user is using the **csh**(1).

mode *type* Ask the remote host for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered. *Type* is either **line** (for "line by line" mode) or **character** (for "character at a time" mode).

status Show the current status of **telnet**. This includes the peer one is connected to, as well as the current mode.

display [*argument...*]
Displays all, or some, of the **set** and **toggle** values (see below).

? [*command*]
Get help. With no arguments, **telnet** prints a help summary. If a command is specified, **telnet** prints the help information for just that command.

send *arguments*
Sends one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

escape Sends the current **telnet** escape character (initially "^]").

synch Sends the **TELNET SYNCH** sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system — if it doesn't work, a lower case "r" may be echoed on the terminal).

brk Sends the **TELNET BRK** (Break) sequence, which may have significance to the remote system.

ip Sends the **TELNET IP** (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

ao Sends the **TELNET AO** (Abort Output) sequence, which should cause the remote system to flush all output from the remote system to the user's terminal.

ayt Sends the **TELNET AYT** (Are You There) sequence, to which the remote system may or may not choose to respond.

ec Sends the **TELNET EC** (Erase Character) sequence, which should cause the remote system to erase the last character entered.

el Sends the **TELNET EL** (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

ga Sends the **TELNET GA** (Go Ahead) sequence, which likely has no significance to the remote system.

nop Sends the **TELNET NOP** (No OPeration) sequence.

? Prints out help information for the **send** command.

set *argument value*

Set any one of a number of **telnet** variables to a specific value. The special value **off** turns off the function associated with the variable. The values of variables may be interrogated with the **display** command. The variables which may be specified are:

echo This is the value (initially ''^E'') which, when in ''line by line'' mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

escape This is the **telnet** escape character (initially ''^['') which causes entry into **telnet** command mode (when connected to a remote system).

interrupt
If **telnet** is in **localchars** mode (see **toggle localchars** below) and the **interrupt** character is typed, a **TELNET IP** sequence (see **send ip** above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's **intr** character.

quit If **telnet** is in **localchars** mode (see **toggle localchars** below) and the **quit** character is typed, a **TELNET BRK** sequence (see **send brk** above) is sent to the remote host. The initial value foı the quit character is taken to be the terminal's **quit** character.

flushoutput
If **telnet** is in **localchars** mode (see **toggle localchars** below and the **flushoutput** character is typed, a **TELNET AC** sequence (see **send ao** above) is sent to the remote host. The initial value for the flush character is taken to be the terminal': **flush** character.

erase If **telnet** is in **localchars** mode (see **toggle localchars** below) **and** if **telnet** is operating in ''character at a time'' mode, theı when this character is typed, a **TELNET EC** sequence (se **send ec** above) is sent to the remote system. The initial valu for the erase character is taken to be the terminal's eras character.

kill If **telnet** is in **localchars** mode (see **toggle localchars** below), **and** if **telnet** is operating in ''character at a time'' mode, then when this character is typed, a **TELNET EL** sequence (see **send el** above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's **kill** character.

eof If **telnet** is operating in ''line by line'' mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's **eof** character.

toggle *arguments...*

Toggle (between TRUE and FALSE) various flags that control how **telnet** responds to events. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are:

localchars

If this is TRUE, then the **flush**, **interrupt**, **quit**, **erase**, and **kill** characters (see **set** above) are recognized locally, and transformed into (hopefully) appropriate **TELNET** control sequences (respectively **ao**, **ip**, **brk**, **ec**, and **el**; see **send** above). The initial value for this toggle is TRUE in ''line by line'' mode, and FALSE in ''character at a time'' mode.

autoflush

If **autoflush** and **localchars** are both TRUE, then when the **ao**, **intr**, or **quit** characters are recognized (and transformed into **TELNET** sequences; see **set** above for details), **telnet** refuses to display any data on the user's terminal until the remote system acknowledges (via a **TELNET** *Timing Mark* option) that it has processed those **TELNET** sequences. The initial value for this toggle is TRUE if the terminal user had not done an "**stty noflsh**", otherwise FALSE (see stty(1)).

autosynch

If **autosynch** and **localchars** are both TRUE, then when either the **intr** or **quit** characters is typed (see **set** above for descriptions of the **intr** and **quit** characters), the resulting **TELNET** sequence sent is followed by the **TELNET SYNCH** sequence. This procedure **should** cause the remote system to begin throwing away all previously typed input until both of the **TELNET** sequences have been read and acted upon. The initial value of this toggle is FALSE.

crmod	Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is FALSE.
debug	Toggles socket level debugging (useful only to the super–user). The initial value for this toggle is FALSE.
options	Toggles the display of some internal **telnet** protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.
netdata	Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.
?	Displays the legal toggle commands.

BUGS

There is no adequate way for dealing with flow control.

On some remote systems, echo has to be turned off manually when in ''line by line'' mode.

There is enough settable state to justify a **.telnetrc** file.

No capability for a **.telnetrc** file is provided.

In ''line by line'' mode, the terminal's **eof** character is only recognized (and sent to the remote system) when it is the first character on a line.

SEE ALSO

init(3N), hosts(4), stty(1)

NAME

test – condition evaluation command

SYNOPSIS

test *expr*

[*expr* **]**

DESCRIPTION

test evaluates the expression *expr* and, if its value is true, sets a zero (true) exit status; otherwise, a non-zero (false) exit status is set; **test** also sets a non-zero exit status if there are no arguments. When permissions are tested, the effective user ID of the process is used.

All operators, flags, and brackets (brackets used as shown in the second SYNOPSIS line) must be separate arguments to the **test** command; normally these items are separated by spaces.

The following primitives are used to construct *expr*:

–r *file*	True if *file* exists and is readable.
–w *file*	True if *file* exists and is writable.
–x *file*	True if *file* exists and is executable.
–f *file*	True if *file* exists and is a regular file.
–d *file*	True if *file* exists and is a directory.
–c *file*	True if *file* exists and is a character special file.
–b *file*	True if *file* exists and is a block special file.
–p *file*	True if *file* exists and is a named pipe (fifo).
–u *file*	True if *file* exists and its set-user-ID bit is set.
–g *file*	True if *file* exists and its set-group-ID bit is set.
–k *file*	True if *file* exists and its sticky bit is set.
–s *file*	True if *file* exists and has a size greater than zero.
–t [*fildes*]	True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
–z *s1*	True if the length of string *s1* is zero.
–n *s1*	True if the length of the string *s1* is non-zero.
–L,**–S**	True if file exists and is a soft link.
s1 **=** *s2*	True if strings *s1* and *s2* are identical.
s1 **!=** *s2*	True if strings *s1* and *s2* are *not* identical.
s1	True if *s1* is *not* the null string.

n1 −eq *n2*	True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **−ne**, **−gt**, **−ge**, **−lt**, and **−le** may be used in place of **−eq**.

These primaries may be combined with the following operators:

!	Unary negation operator.
−a	Binary *and* operator.
−o	Binary *or* operator (**−a** has higher precedence than **−o**).
(expr)	Parentheses for grouping. Notice also that parentheses are meaningful to the shell and, therefore, must be quoted.

WARNING

If you test a file you own (the **−r**, **−w**, or **−x** tests), but the permission tested does not have the *owner* bit set, a non-zero (false) exit status will be returned even though the file may have the *group* or *other* bit set for that permission. The correct exit status will be set if you are super-user.

The = and != operators have a higher precedence than the **−r** through **−n** operators, and = and != always expect arguments; therefore, = and != cannot be used with the **−r** through **−n** operators.

If more than one argument follows the **−r** through **−n** operators, only the first argument is examined; the others are ignored, unless a **−a** or a **−o** is the second argument.

SEE ALSO

find(1), sh(1).

NAME

tftp – trivial file transfer protocol

SYNOPSIS

tftp [**–g|g!|p|r|w**] *localname host foreignname* [*mode*]

DESCRIPTION

tftp is the front-end to the Trivial File Transfer Protocol. It enables you to copy files among internet hosts without remote user-level access. A minus sign (–) may be substituted for *localname* in which case the standard input (or output) will be used.

tftp requires a switch to dictate the direction of the file transfer. The recognized switches are:

put (–p, –w)	Write the local file (*localname*) onto the foreign host's file system as *foreignname*. Note that *foreignname* must be quoted if it contains shell special characters. (The word **put**, the switch **–p**, and the switch –w are all synonymous).
get (–g, –r)	Read the foreign host's file (*foreignname*) into the local file, *localname*. If *localname* already exists, **tftp** will fail with an appropriate error message.
get! (–g!)	Perform a **tftp get**, overwriting the local file (if it exists). Note that the exclamation point following the command indicates that the command will modify a data structure (in this case, it will overwrite an existing file; the syntax is derived from the Yale/T and MIT/Scheme naming conventions). Within a UNIX shell, the exclamation point must be escaped (usually with a backslash) to avoid shell interpretation.

TRANSFER MODES

Mode may be **netascii,** or **image. netascii**, the default mode, transfers the file as standard ascii characters. Image mode transfers the file in binary, with no character conversion.

NOTES

The Domain/OS SysV versions of **tftp** and **tftpd**(1M) are adaptations of the MIT Project Athena implementations of the **tftp** protocol. Domain/OS SysV **tftp** will interface with any RFC783 compliant implementation. Note, however, that the 4.3BSD distribution version of **tftp** does not meet these restrictions.

WARNINGS

tftp and **tftpd**(1M) comprise an implementation of the Trivial File Transfer Protocol described in RFC783. They allow you to quickly copy files among hosts on an internet without regard to ownership or access restrictions. Therefore, the desired security of a system should be considered before allowing **tftp** transactions. In an inspired attempt to prevent accidental destruction of important files, **tftp** requires that remote file names be absolute pathnames (that is, beginning with /) containing the string "/tftp/", but not containing the string "/../".

EXAMPLES

Each of the following examples presumes that there is a host on the internet called **carrara**, running a **tftp** server.

To copy the local file /tftp/foo to **carrara**, and deposit it in **carrara**'s /tftp directory under the name **bar**:

```
tftp -p /tftp/foo carrara /tftp/bar
```

To copy the remote file (on **carrara**) named /tftp/bar to the local file named /tftp/new:

```
tftp get /tftp/new carrara /tftp/bar
```

To coy the remote binary file (on **carrara**) named /tftp/zed to the local file named /tftp/new, overwriting the old copy of /tftp/new:

```
tftp -g\! /tftp/new carrara /tftp/zed image
```

SEE ALSO

tftpd(1M)
Configuring and Managing TCP/IP.

NAME

time – time a command

SYNOPSIS

time *command*

DESCRIPTION

Once a *command* has executed, **time** prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on standard error.

SEE ALSO

times(2) in the *SysV Programmer's Reference*.

NAME

timex – time a command; report process data and system activity

SYNOPSIS

timex [*options*] *command*

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported. The output of **timex** is written on standard error.

OPTIONS

−p List process accounting records for *command* and all its children. Suboptions **f**, **h**, **k**, **m**, **r**, and t modify the data items reported. The options are as follows:

- **−f** Print the *fork*/*exec* flag and system exit status columns in the output.
- **−h** Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This ''hog factor'' is computed as:

  (*total CPU time*)/(*elapsed time*).

- **−k** Instead of memory size, show total kcore-minutes.
- **−m** Show mean core size (the default).
- **−r** Show CPU factor (*user time*/(*system-time* + *user-time*).
- **−t** Show separate system and user CPU times. The number of blocks read or written and the number of characters transferred are always reported.

−o Report the total number of blocks read or written and total characters transferred by *command* and all its children.

−s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in **sar**(1) are reported.

WARNING

Process records associated with *command* are selected from the accounting file /**usr**/**adm**/**pacct** by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

EXAMPLES

A simple example:

```
timex -ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
timex -opskmt sh
        session commands
EOT
```

SEE ALSO

sar(1).

NAME

touch – update access and modification times of a file

SYNOPSIS

touch [**–amc**] [mmddhhmm[yy]] *files*

DESCRIPTION

touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see **date**(1)) the current time is used. The **–a** and **–m** options cause **touch** to update only the access or modification times respectively (default is **–am**). The **–c** option silently prevents **touch** from creating the file if it did not previously exist.

The return code from **touch** is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1).
utime(2) in the *SysV Programmer's Reference*.

NAME

tplot – graphics filters

SYNOPSIS

tplot [**–T***terminal* [**–e** *raster*]]

DESCRIPTION

These commands read plotting instructions (see **plot**(4)) from the standard input and in general produce, on the standard output, plotting instructions suitable for a particular *terminal*. If no *terminal* is specified, the environment parameter **$TERM** (see **environ**(5)) is used. Known *terminal*s are:

300	DASI 300.
300S	DASI 300s.
450	DASI 450.
4014	Tektronix 4014.
ver	Versatec D1200A. This version of **plot**(4) places a scan-converted image in **/usr/tmp/raster$$** and sends the result directly to the plotter device, rather than to the standard output. The **–e** option causes a previously scan-converted file *raster* to be sent to the plotter.

FILES

/usr/lib/t300
/usr/lib/t300s
/usr/lib/t450
/usr/lib/t4014
/usr/lib/vplot
/usr/tmp/raster$$

SEE ALSO

plot(3X), plot(4), term(5) in the *SysV Programmer's Reference*.

NAME

tpm – set/display touchpad and mouse characteristics

SYNOPSIS

tpm [*options*]

DESCRIPTION

tpm allows you to define characteristics for the touchpad and mouse. The touchpad operates in one of three modes: absolute, relative, and absolute/relative. The mode of operation establishes how movements of your finger on the touchpad affect the position of the cursor on the screen. The three modes differ primarily in how the cursor moves when you lift your finger from the touchpad and then replace it. The mouse operates in relative mode only, and –s is the only valid option.

The subsections below describe the three operational modes, as well as the other options.

OPTIONS

If no options are specified, **tpm** displays the current touchpad characteristics.

–a (default)	sSelect absolute mode.
–r	Selects relative mode.
–ar	Selects absolute/relative mode.
–rerange	Sets prescaling factors for touchpad data.
–s *x y*	Sets scaling factors for *x* and *y*. Values can range from –32768 to 32767. The default scaling factors are 799 for *x* and 1023 for *y* (portrait displays); and 1023 for *x* and 799 for *y* (landscape displays).
–o *x y*	Sets *x* and *y* as the origin for absolute mode. Values must be in raster units, and can range from 0 to 1023. The default origin is 0,0.
–h *n*	Sets the hysteresis box size. The value must be in raster units, and can range from 0 to 1023. The default is 5.

DESCRIPTION

Absolute Mode

In absolute mode, using the default scale and origin, the touchpad approximates the screen, so that the top left edge of the touchpad represents cursor positions at the top left edge of the screen. Absolute mode is the default setting. When you place your finger on the touchpad, the cursor jumps to a corresponding position on the screen. Moving your finger across the touchpad moves the cursor across the screen in the same direction.

For example, moving your finger from the top of the touchpad to the bottom moves the cursor from top to bottom on the screen. If you lift your finger from the touchpad, and later touch the pad again, the cursor jumps to a new position on the screen corresponding to the new finger position.

Relative Mode

In relative mode, cursor movements correspond only to finger movements across the touchpad. The cursor does not move when you first place your finger on the touchpad. This differs from absolute mode, where the cursor jumps to a new position when you lift your finger and then replace it. In effect, relative mode causes the touchpad to correspond to different areas of the screen, relative to the current cursor position.

This is the only meaningful mode for a mouse: all movement begins from the current cursor position.

Relative mode is typically used with scale factors less than the defaults. Smaller scale factors mean that the touchpad maps to a smaller area of the screen. For example, scale factors of 200 by 256 specify one-sixteenth of a portrait display's screen area. With small scale factors, relative mode allows fine resolution of the cursor position within a small area.

To reach distant areas on the screen, you can use several strokes on the touchpad or mouse, each stroke moving the cursor closer to its final destination. To assist you in making large movements in relative mode without having to use too many strokes, the speed of cursor movement is artificially accelerated in relation to the speed of finger or mouse movement. Thus, a quick motion moves the cursor farther than a slow, deliberate motion which covers the same distance.

Absolute/Relative Mode

Absolute/relative mode is a combination of absolute and relative modes. It has no meaning for the mouse. In this mode, the first position of your finger on the touchpad establishes the first position of the cursor, as in absolute mode. Moving your finger across the touchpad moves the cursor across the screen. As in relative mode, the scale is typically smaller than the whole screen.

Unlike absolute and relative modes, however, the effect of lifting your finger from the touchpad depends on how long you break contact. If you lift and replace your finger quickly — within a half second — the cursor does not move, and the effect is the same as relative mode. If you break contact for more than a half second, however, the cursor jumps to a new absolute position when you put your finger on the touchpad again.

Absolute/relative mode is useful for jumping the cursor from one place to another, then carefully positioning it in the new area. For example, this mode is commonly used to move the cursor in a jump from one window to another, and then point to a character in the second window.

Prescaling the Touchpad

Raw touchpad data varies slightly from one touchpad to another. Prescaling is, in essence, calibration of the touchpad. Every time you start the node, the touchpad manager prescales the data to determine an exact range for the device.

To prescale, the touchpad manager observes the first thousand points of touchpad data (about 30 seconds of use). During this time, you should try to touch all four edges of the touchpad to ensure that the observed data constitutes an accurate sample. Based on the observed data, the touchpad manager computes a prescaling factor which, when applied to the data, brings all points into the range from –.05 to 1.05. This range corresponds to the edges of the screen, plus an overlap of 5%, when multiplied by the default scaling factors. Because of the overlap, you need not touch the internal frame (under the conductive material) to move the cursor to the edge of the screen.

The **–rerange** option invokes prescaling. This option is useful if the first 30 seconds of use did not include the entire range of the touchpad. It is also handy if you change keyboards on a node, and therefore need to reset the prescaling factors without restarting the node.

Scale Factors

The touchpad manager translates, or scales, the data into raster units, which the Display Manager understands. Scale factors, specified with the –s option, are applied to the prescaled touchpad data to translate it to raster units for the Display Manager.

The scale factors are multiplied by the prescaled data. The default scale factors are 800 for x and 1024 for y (portrait displays); and 1024 for x and 800 for y (landscape displays). Applying these factors to prescaled data results in numbers from approximately 0 to 799 (for x) and 0 to 1023 (for y) for portrait displays, and vice versa for landscape displays. (Note that the prescaled data allows a 5% overlap, as mentioned in the preceding subsection.)

The default scale factors provide for touchpad data corresponding to the whole screen. Smaller scale factors reduce the area to which the touchpad maps, thereby allowing you to more finely tune the cursor position. This also applies to mouse movement, allowing changes in the apparent motion sensitivity of the device.

Setting the Origin

The origin is the point denoted by the upper left corner of the touchpad, in absolute and absolute/relative mode. In relative mode, the origin has no meaning. By default, the touchpad origin corresponds to the upper left corner of the screen, that is, the point 0,0 in raster units. By changing the origin, you can use the touchpad (in absolute mode) to correspond to a portion of the screen.

This feature is useful for applications that need to move the cursor within a fixed window, rather than across the whole screen. For example, a program that displays a menu in one window might set the origin to the upper left corner of the menu window. Consequently, the touchpad maps onto the menu window instead of the entire screen.

Hysteresis

The hysteresis value defines the dimensions of a box around your finger position on the touchpad or the current position of the mouse. Movement within the box does not change the position of the cursor on the screen.

Specify the hysteresis value in raster units. The touchpad manager compares the value to the difference between the current and previous finger positions on the touchpad or the current and previous positions of the mouse. If the difference is less than the hysteresis value, the cursor does not move. If the difference is greater than the hysteresis value, the hysteresis value is subtracted from the difference and the cursor moves the resulting distance. The default hysteresis value is five.

EXAMPLES

Display current characteristics.

```
$ tpm
  Mode: absolute
  Xscale: 1024, Yscale: 800
  Hysteresis: 5
  Origin: 0, 0
```

Set characteristics to absolute/relative mode with half the default scaling sensitivity (portrait display).

```
$ tpm -ar -s 400 512
```

NAME

tput – initialize a terminal or query terminfo database

SYNOPSIS

tput [–Ttype] *capname* [*parms* ...]

tput [–Ttype] **init**

tput [–Ttype] **reset**

tput [–Ttype] **longname**

DESCRIPTION

tput uses the **terminfo**(4) database to make the values of terminal-dependent capabilities and information available to the shell (see **sh**(1)), to initialize or reset the terminal, or return the long name of the requested terminal type. **tput** outputs a string if the attribute (*cap*ability *name*) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, **tput** simply sets the exit code (**0** for TRUE if the terminal has the capability, **1** for FALSE if it does not), and produces no output. Before using a value returned on standard output, the user should test the exit code ($?, see **sh**(1)) to be sure it is **0**. (See **EXIT CODES** and **DIAGNOSTICS** below.) For a complete list of capabilities and the *capname* associated with each, see **terminfo**(4). This has no effect on Apollo transcript pads. It is useful on connected terminals and VT100 windows.

OPTIONS

–T*type* indicates the *type* of terminal. Normally this option is unnecessary, because the default is taken from the environment variable **TERM.** If –T is specified, then the shell variables **LINES** and **COLUMNS** and the layer size will not be referenced.

capname indicates the attribute from the **terminfo**(4) database.

parms If the attribute is a string that takes parameters, the arguments *parms* will be instantiated into the string. An all numeric argument will be passed to the attribute as a number.

init If the **terminfo**(4) database is present and an entry for the user's terminal exists (see –T*type,* above), the following will occur: (1) if present, the terminal's initialization strings will be output (**is1**, **is2**, **is3**, **if**, **iprog**), (2) any delays (e.g., newline) specified in the entry will be set in the tty driver, (3) tabs expansion will be turned on or off according to the specification in the entry, and (4) if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will silently be skipped.

reset Instead of putting out initialization strings, the terminal's reset strings will be output if present (**rs1**, **rs2**, **rs3**, **rf**). If the reset strings are not present, but initialization strings are, the initialization strings will be output.

Otherwise, **reset** acts identically to **init**.

longname If the **terminfo**(4) database is present and an entry for the user's terminal exists (see −T*type* above), then the long name of the terminal will be put out. The long name is the last name in the first line of the terminal's description in the **terminfo**(4) database (see **term**(5)).

EXAMPLES

tput init	Initialize the terminal according to the type of terminal in the environmental variable **TERM**. This command should be included in everyone's **.profile** after the environmental variable **TERM** has been exported, as illustrated on the **profile**(4) manual page.
tput −T5620 reset	Reset an AT&T 5620 terminal, overriding the type of terminal in the environmental variable **TERM**.
tput cup 0 0	Send the sequence to move the cursor to row **0**, column **0** (the upper left corner of the screen, usually known as the "home" cursor position).
tput clear	Echo the clear-screen sequence for the current terminal.
tput cols	Print the number of columns for the current terminal.
tput -T450 cols	Print the number of columns for the 450 terminal.
bold=‘tput smso‘	
offbold=‘tput rmso‘	Set the shell variables **bold**, to begin stand-out mode sequence, and **offbold**, to end standout mode sequence, for the current terminal. This might be followed by a prompt: **echo "${bold}Please type in your name: ${offbold}\c"**
tput hc	Set exit code to indicate if the current terminal is a hardcopy terminal.
tput cup 23 4	Send the sequence to move the cursor to row 23, column 4.
tput longname	Print the long name from the **terminfo**(4) database for the type of terminal specified in the environmental variable **TERM**.

EXIT CODES

If *capname* is of type boolean, a value of **0** is set for TRUE and **1** for FALSE. If *capname* is of type string, a value of **0** is set if the *capname* is defined for this terminal *type* (the value of *capname* is returned on standard output); a value of **1** is set if *capname* is not defined for this terminal *type* (a null value is returned on standard output).

If *capname* is of type integer, a value of **0** is always set, whether or not *capname* is defined for this terminal *type*. To determine if *capname* is defined for this terminal *type*, the user must test the value of standard output. A value of **−1** means that *capname* is not defined for this terminal *type*. Any other exit code indicates an error; see

BR DIAGNOSTICS ,

FILES

/usr/lib/terminfo/?/*	Compiled terminal description database
/usr/include/curses.h	**curses**(3X) header file
/usr/include/term.h	**terminfo**(4) header file
/usr/lib/tabset/*	Tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs); for more information, see **terminfo**(4).

DIAGNOSTICS

tput prints the following error messages and sets the corresponding exit codes.

Exit Code	Error Message
0	−1 (*capname* is a numeric variable that is not specified in the **terminfo**(4) database for this terminal type, e.g., **tput −T450 lines** and **tput −T2621 xmc**)
1	no error message is printed, see **EXIT CODES**, above.
2	usage error
3	unknown terminal *type* or no *terminfo*(4) database
4	unknown **terminfo**(4) capability *capname*

SEE ALSO

stty (1), tabs (1).
profile(4), terminfo(4) in the *SysV Programmer's Reference*.

NAME

tr – translate characters

SYNOPSIS

tr [**–cds**] [*string1* [*string2*]]

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **–cds** may be used.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the *strings*:

[a–z]	Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.
[a**n*]	Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

OPTIONS

–c	Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.
–d	Deletes all input characters in *string1*.
–s	Squeezes all strings of repeated output characters that are in *string2* to single characters.

EXAMPLE

The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

```
tr –cs "[A–Z][a–z]" "[\012*]" <file1 >file2
```

BUGS

Will not handle ASCII **NUL** in *string1* or *string2*; always deletes **NUL** from input.

SEE ALSO

ed(1), sh(1).
ascii(5) in the *SysV Programmer's Reference*.

NAME

tr_font – transliterate characters within a font

SYNOPSIS

tr_font *font_name hex_conversion_table*

DESCRIPTION

The **tr_font** command allows you to change the order in which characters appear in fonts. It rearranges the graphic images associated with the characters in *font_name*, according to information in the *hex_conversion_table*. Use it if you create a new font file from two font files that have different character orders.

tr_font only works on fonts already formatted for SR10. It works directly on the font, without creating a backup. The format for the hex_conversion_table file is:

```
src_ordinal dest_ordinal comment
src_ordinal dest_ordinal comment
src_ordinal dest_ordinal comment
```

where *src_ordinal* is the hexidecimal ordinal value of the character whose graphic image is to be used as the source, *dest_ordinal* is the ordinal value of the character which gets the transliterated image, and comment is an optional remark (for the ASCII character set, the hexidecmal ordinal value 41 represents the character 'A'). If the font was created by concatenating two fonts with **cvt_font**, then the hexidecimal ordinal value of the lowest possible character in the second font is 80.

EXAMPLE

The following example rearranges the characters in the SR10 format font file named **courier** according to the information in the *hex_conversion_table* **theirs_to_ours**.

```
$ tr_font courier theirs_to_ours
```

This is a sample of a *hex_conversion_table* file.

```
A1 A1 !down
A2 A2 cent
A3 A3 sterling
A5 A5 yen
A7 A7 section
A8 A4 currency
AB AB guillemotleft
B6 B6 paragraph
```

```
B7 B7 bullet
B8 B8 quitesinglebase
BB BB guill
```

SEE ALSO

cvt_font(1)

NAME

trconf – list active Streams or configure STREAMS trace modules

SYNOPSIS

trconf [**–l**] [**–i** *streamid moduleno name*]
[**–r** *streamid moduleno*]
[**–c** *name mtype dir onoff timestamp verbiage*]
[**–a** *name trmon*]
[**–p** *name pattern*]

DESCRIPTION

trconf is the trace module (**tracem**(7)) configuration utility. It allows you to list currently active Streams in the system, to insert trace module instances into any Stream, remove trace module instances from any Stream, set parameters of any trace module instance, associate a tracing instance of the trace module with a reporting instance (created by a **trmon** invocation), or set a pattern for the pattern matching function of a trace module instance.

The push functionality has been extended for the trace module to allow insertion into Streams at any point below the Stream head and above the Stream tail (driver). The exception is a lower multiplexed Stream, where insertion is only allowed below the linked module and above the Stream tail. To overcome this limitation, push the **nulm** module onto the lower Stream, prior to linking the Stream under the multiplexor. The desired tracing configuration can then be achieved by inserting **tracem** under the **nulm** module.

OPTIONS

–l Lists queues in each active Stream.

–i *streamid moduleno name*
Insert a trace module instance into a Stream above a module. *streamid* is a Stream index as returned by the **–l** option. *moduleno* is the index of the module within the Stream. *name* is a string with the name for this trace module instance.

–r *streamid moduleno*
Remove a trace module instance from a Stream. *streamid* is a Stream index as returned by the **–l** option. *moduleno* is the index of the trace module instance within the Stream.

–c *name mtype dir onoff timestamp verbiage*
Configure the attributes of a trace module instance. *name* is a string with the name for this trace module instance. *mtype* is a vertical bar (|) separated list of message types. (The vertical bar must be quoted to prevent the shell from interpreting it as a pipe.) *dir* is a | separated list of up or down. *onoff* is either on or off. *timestamp* is either accurate, inaccurate or off. *verbiage* is an integer from 0 to 7 (ignored).

–a *name trmon*
: Associate a trace module instance with the **trmon** instance that is to output the messages being traced. *name* is a string with the name for this trace module instance. *trmon* is a string with a name for the **trmon** instance.

–p *name pattern*
: Set a pattern for the pattern matching function of a trace module instance. *name* is a string with the name for this trace module instance. *pattern* is a string containing 0's, 1's and x's (x matches 0 or 1).

EXAMPLES

List the active Streams.

```
$ trconf –l
Stream index: 0
Queue  0: strwhead
Queue  1: LOG
```

Insert a **tracem** instance named *traceml* on top of the LOG module above.

```
$ trconf –i 0 1 traceml
```

Set up parameters of this instance.

```
$ trconf –c traceml m_data '|' m_proto up on accurate 7
```

Enable pattern matching.

```
$ trconf –p traceml 1010xxxx
```

Assign the **trmon** instance *trmonl* to the **tracem** instance *traceml*. This assumes the **trmon** utility has already been used to create the **trmon** instance called *trmonl*. All selected messages passing through *traceml* will be reported by *trmonl*.

```
$ trconf –a traceml trmonl
```

When done tracing, remove the **tracem** instance.

```
$ trconf –r 0 1
```

SEE ALSO

trmon(1), strinfo(1)

NAME

trmon – print messages collected by trace modules on active Streams

SYNOPSIS

trmon [**–h**] [**–n** *name*] [**–f** *format_file*]

DESCRIPTION

trmon prints STREAMS messages collected by instances of **tracem** modules pushed onto Streams with **trconf**. For each reported message, a header containing reporting instance id, sequence id, message type, timing information, and message direction is printed. The header is followed (if **–f** is used) by the contents of the data part of the message.

OPTIONS

–h
: Suppress printing of header output.

–n *name*
: Give this **trmon** instance a name. *name* is an alpha-numeric string. If no *name* is given, the **trmon** instance will have the name **trmon***n*, where *n* is its internal id.

–f *format_file*
: Use *format_file* to specify the format to print the data part of the reported messages. This file is in ''modified C struct syntax.'' Customized output formats for any type of message can be formed by taking structure definitions from the header file of a protocol and making some minor modifications. The format file consists of a number of lines for each data field to be recognized as follows:

string type [*fieldname*] [*comment*]

where:

string	is a string delimited by double-quotes ("") and containing a **printf** format string, or a null string ("") meaning don't print this field.
type	is a C basic type, or **dump**, which produces a hexadecimal dump of the rest of the message (like **od**(1) **–h**), or **raw**, which outputs the raw bytes.
fieldname	is a C field identifier and is optional and ignored.
comment	is a C comment and is optional and ignored.

EXAMPLES

The invocation:

```
trmon -f dump_file
```

where *dump_file* is:

```
" "      dump a; /* to dump the entire message */
```

will cause **trmon** to choose its own name, and dump all messages it receives from all **tracem** instances.

The invocation:

```
trmon -h -n trace_ioctl -f ioc_file
```

where **ioc_file** is:

```
"cmd: %d,"       int              ioc_cmd;     /* ioctl command */
" uid: %u,"      unsigned short   ioc_uid;     /* effective uid */
" gid: %u,"      unsigned short   ioc_gid;     /* effective gid */
" id: %u,"       unsigned int     ioc_id;      /* ioctl id */
" count: %u,"    unsigned int     ioc_count;   /* # bytes of data */
" error: %d,"    int              ioc_error;   /* error code */
" rval: %d\n"    int              ioc_rval;    /* return value  */
" "              dump             a;           /* dump data */
```

will cause **trmon** to name itself **trace_ioctl** and to format all messages it receives, without header information. In this case, it is assumed that **trconf** will be used to configure a **tracem** instance that reports only on M_IOCTL messages, and to associate that tracing instance with this **trmon** instance.

SEE ALSO

trconf(1), tracem(7), strinfo(1)

NAME

true, **false** – provide truth values

SYNOPSIS

true

false

DESCRIPTION

true does nothing, successfully. **false** does nothing, unsuccessfully. They are typically used in input to **sh**(1) such as:

```
while true
do
        command
done
```

DIAGNOSTICS

true has exit status zero; **false** has exit status nonzero.

SEE ALSO

sh(1).

NAME

ts – display the module name and time stamp

SYNOPSIS

ts [–nhd] *object_module_name*

DESCRIPTION

ts displays the time stamp and module name stored in an object module. Shown is the time and date that the module was created by one of the linkers or compilers. The time stamp is not affected by copying an object module, so it is a reliable indicator of whether particular object modules are copies of one another.

OPTIONS

–nhd Option does not print the table header. ts outputs in tabular format with table header by default.

NAME

tsort – topological sort

SYNOPSIS

tsort [*file*]

DESCRIPTION

The **tsort** command produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

SEE ALSO

lorder(1).

NAME

tty – get the name of the terminal

SYNOPSIS

tty [–s]

DESCRIPTION

Tty prints the pathname of your terminal.

OPTIONS

–s	Inhibit printing of the terminal pathname, allowing testing of the exit code only.
–l	Not supported.

EXIT CODES

2	if invalid options were specified
0	if standard input is a terminal
1	otherwise

DIAGNOSTICS

not a tty	Standard input is not a terminal and –s is not specified.

NAME

tz – set or display system time zone

SYNOPSIS

tz [*tz_name* | *utc_delta* [*new_tz*]]

DESCRIPTION

tz sets the system time zone to a known time zone or to an offset from Coordinate Universal Time (utc). If no arguments are specified, **tz** displays the current setting.

tz_name (optional) Specify new time zone. The following are valid names:

Name	Time Zone
EDT	Eastern Daylight Time
EST	Eastern Standard Time
CDT	Central Daylight Time
CST	Central Standard Time
MDT	Mountain Daylight Time
MST	Mountain Standard Time
PDT	Pacific Daylight Time
PST	Pacific Standard Time
GMT	Greenwich Mean Time
UTC	Coordinated Universal Time

TILDE ESCAPES.if 0=0 .nr c. 54565-0-14

Default if omitted: use *utc_delta* argument

utc_delta (optional) Specify positive or negative offset from utc. The plus sign is optional for positive offsets. Format for offset is *hh:mm* (for example, −10:00 for ten hours earlier than, west of, Coordinated Universal Time). Only whole or half hour offsets may be specified. Other fractional offsets produce an error message.

Default if omitted: use *tz_name* argument

new_tz (optional) Specify new time zone name to be assigned to the zone indicated by the *utc_delta* argument. Use this argument to create time zones that are not included in the list above.

Default if omitted: no name assigned

EXAMPLES

Display current time zone.

```
$ tz
  Timezone:       EST
  Delta from UTC: -5:00
```

Set time zone to Pacific Daylight Time.

```
$ tz pdt
```

Create (and set) a time zone named GST that is four and a half hours later than (east of) Coordinated Universal Time.

```
$ tz 4:30 gst
```

NAME

uk_to_iso – convert files to ISO format

SYNOPSIS

uk_to_iso *input_file output_file*

DESCRIPTION

These utilities convert files written with the overloaded 7-bit national fonts to the Internation Standards Organization (ISO) 8-bit format. The overloaded fonts include any with a specific language suffix (for example, **f7x13.french**, or **din_f7x11.german**). If you created and/or edited a file using one of the national fonts, that file is a candidate for conversion.

You are not required to convert files, but probably will want to because

1. The overloaded fonts replace certain ASCII characters with national ones, have that subset of ASCII characters and the national characters in one file. The 8-bit fonts available as of SR10 include all the ASCII characters and the national characters.
2. The 8-bit fonts also include a wider range of national characters, so you can enter any character in any western European language. This was not always possible with the overloaded fonts. For example, there was not enough space in the overloaded font to include all the French characters, but they all exist in the 8-bit fonts.

There are two parameters to the conversion utilities. You must provide the name of the file you want to convert (*input_file*) and your *output_file*. If *output_file* already exists, the utilities abort.

The default location for the utilities is **/usr/apollo/bin**.

FILES

/usr/apollo/bin/french_to_iso	Converts overloaded French to ISO format
/usr/apollo/bin/german_to_iso	Converts overloaded German to ISO format
/usr/apollo/bin/nor.dan_to_iso	Converts overloaded Norwegian/Danish to ISO format
/usr/apollo/bin/swedish_to_iso	Converts overloaded Swedish/Finnish to ISO format
/usr/apollo/bin/swiss_to_iso	Converts overloaded Swiss to ISO format
/usr/apollo/bin/uk_to_iso	Converts overloaded U.K. English to ISO format

DIAGNOSTICS

All messages are generally self-explanatory.

NAME

umask – set file-creation mode mask

SYNOPSIS

umask [*ooo*]

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see **chmod**(2) and **umask**(2)). The value of each specified digit is subtracted from the corresponding ''digit'' specified by the system for the creation of a file (see **creat**(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode 777 become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

umask is recognized and executed by the shell.

umask can be included in your **.profile** (see **profile**(4)) and invoked at login to automatically set your permissions on files or directories created.

SEE ALSO

chmod(1), sh(1).
chmod(2), creat(2), umask(2), profile(4) in the *SysV Programmer's Reference*.

NAME

uname – print name of current UNIX system

SYNOPSIS

uname [−snrvma]
uname [−S system name]

DESCRIPTION

uname prints the current system name of the UNIX system on the standard output file. It is mainly useful to determine which system one is using.

OPTIONS

−s	prints the system name (default).
−n	Prints the nodename (the nodename is the name by which the system is known to a communications network).
−r	Prints the operating system release.
−v	Prints the operating system version.
−m	Prints the machine hardware name.
−a	Prints all the above information.

SEE ALSO

uname(2) in the *SysV Programmer's Reference*.

NAME

unget – undo a previous get of an SCCS file

SYNOPSIS

unget [–rSID] [–s] [–n] *files*

DESCRIPTION

unget undoes the effect of a **get –e** done prior to creating the intended new delta. If a directory is named, **unget** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

OPTIONS

–r*SID* Uniquely identifies which delta is no longer intended. (This would have been specified by **get**(1) as the "new delta"). The use of this keyletter is necessary only if two or more outstanding gets for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

–s Suppresses the printout, on the standard output, of the intended delta's *SID*.

–n Causes the retention of the gotten file which would normally be removed from the current directory.

SEE ALSO

delta(1), get(1), sact(1).

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [**–udc** [**+n**] [**–n**]] [*input* [*output*]]

DESCRIPTION

uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see **sort**(1). If the **–u** flag is used, just the lines that are not repeated in the original file are output. The **–d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **–u** and **–d** mode outputs.

The **–c** option supersedes **–u** and **–d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

–n The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+n The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

comm(1), sort(1).

NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have:  inch
You want:  cm
           * 2.540000e+00
           / 3.937008e–01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have:  15 lbs force/in2
You want:  atm
           * 1.020689e+00
           / 9.797299e–01
```

units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	Ratio of circumference to diameter,
c	Speed of light,
e	Charge on an electron,
g	Acceleration of gravity,
force	Same as g,
mole	Avogadro's number,
water	Pressure head per unit height of water,
au	Astronomical unit.

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

FILES

/usr/lib/unittab

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [**–f**] *name* ...

pcat *name* ...

unpack *name* ...

DESCRIPTION

Pack attempts to store the specified files in a compressed form. Wherever possible and useful, it replaces each input file *name* by a packed file *name.z* with the same access modes, access and modified dates, and owner as those of *name*.

If **pack** is successful, it removes *name*. Packed files can be restored to their original form using **unpack** or **pcat**.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each *.z* file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75 percent of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90 percent of the original size. **Pack** returns a value equaling the number of files not compressed.

No packing occurs if one or more of the following conditions exists:

the file appears to be already packed
the filename has more than 12 characters
the file has links
the file is a directory
the file cannot be opened
no disk storage blocks will be saved by packing
a file called *name.z* already exists
the *.z* file cannot be created
an I/O error occurred during processing.

The last segment of the filename must contain no more than 12 characters to allow space for the appended *.z* extension. Directories cannot be compressed.

Pcat does for packed files what **cat**(1) does for ordinary files, except that **pcat** cannot be used as a filter. The specified files are unpacked and written to the standard output. To view a packed file named *name.z* use:

pcat *name.z*

or just:

pcat *name*

To make an unpacked copy, say *nnn*, of a packed file named *name.z* (without destroying *name.z*) use the command:

pcat name > nnn

Pcat returns the number of files it was unable to unpack, but will fail if one of the following conditions exist:

the filename (exclusive of the *.z*) has more than 12 characters
the file cannot be opened
the file does not appear to be the output of **pack**.

Unpack expands files created by **pack**. For each file *name* specified in the command, a search is made for a file called *name.z* (or just *name*, if *name* ends in *.z*). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the *.z* suffix stripped from its name. It also has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. It will fail for the same reasons as those listed for **pcat**, as well as for the following additional reasons:

a file with the ''unpacked'' name already exists
the unpacked file cannot be created.

OPTIONS

(Note that these options are only for use with **pack**.)

– Set an internal flag that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* cause the internal flag to be set and reset.

–f Force packing of *name*. Useful for causing an entire directory to be packed even if some of the files will not benefit.

NOTES TO SysV USERS

The Apollo version of the **pack** command creates packed files that have an Apollo file type of ''uasc''. The original file type information is not carried over to the packed file. The **unpack** command checks the magic number of the unpacked file. If it matches one of the Apollo object types or archive type, the file type of the unpacked file is changed from ''uasc'' to ''obj''. If the file type of the original file is other than ''uasc'' or one of the ''obj'' types checked for by **unpack**, the file type must be manually changed after the file is unpacked.

SEE ALSO

cat(1).

NAME

uucp, uulog, uuname – UNIX-to-UNIX system copy

SYNOPSIS

uucp [*options*] *source-files destination-file*

uulog [*options*] **–s***system*
uulog [*options*] *system*
uulog [*options*] **–f***system*

uuname [**–l**] [**–c**]

DESCRIPTION

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A filename may be a pathname on a machine, or may have the following form:

system name!*pathname*

where *system name* is taken from a list of system names that **uucp** knows about. The *system name* may also be a list of names such as:

system name!*system name*!...!*system name*!*pathname*

in which case an attempt is made to send the file via the specified route, to the destination. See NOTES and BUGS below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to forward information.

The question mark (?), asterisk (*), and bracketed ellipsis ([...]) Shell metacharacters appearing in *pathname* are expanded on the appropriate system.

Pathnames may be one of the following (anything else is prefixed by the current directory): A full pathname A pathname preceded by ˜*user* where *user* is a log-in name on the specified system and is replaced by that user's log-in directory A path name preceded by ˜/*destination* where *destination* is appended to **/usr/spool/uucppublic**. This destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a slash mark (/). For example, ˜**/dan/** as the destination will make the directory **/usr/spool/uucppublic/dan** if it does not exist and put the requested file(s) in that directory). <.PP If the result is an erroneous pathname for the remote system, the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions. See **chmod**(2) for more information about permissions. All files received by **uucp** will be owned by **uucp**.

uulog queries a summary log of **uucp** or **uuxqt** transactions in the files **/usr/spool/uucp/.Log/uucico/system**, or **/usr/spool/uucp/.Log/uuxqt/system.**

uuname lists the names of systems known to **uucp.**

The DOMAIN/OS version of **uucp** supports the Vadic Autodialer.

OPTIONS

uucp options

The following options are interpreted by **uucp** *only* :

–c	Does not copy the local file to the spool directory for transfer to the remote machine (default).
–C	Forces the copy of local files to the spool directory for transfer.
–d	Makes all necessary directories for the file copy (default).
–f	Does not make intermediate directories for the file copy.
–g*grade*	*Grade* is a single letter/number; lower ascii sequence characters cause the job to be transmitted earlier during a particular conversation.
–j	Output the job identification ASCII string on the standard output. This job identification can be used by **uustat** to obtain the status or terminate a job.
–m	Send mail to the requester when the copy is completed. The **–m** option only works sending files or receiving a single file. Receiving multiple files specified by special shell characters ? * [...] will not activate the **–m** option.
–n*user*	Notifies you on the remote system that a file was sent.
–r	Does not start the file transfer, just queuse the job.
–s*file*	Reports status of the transfer to *file*. Notes that the *file* must be a full path name.
–x*debug_level*	Produces debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information. (Debugging will not be available if **uucp** was compiled with -DSMALL.)

uulog options

The following options are used by **uulog** *only;* they cause **uulog** to print logging information:

–s*sys*	Prints information about file transfer work involving system *sys*.
–f*system*	Does a **tail** –f of the file transfer log for *system*. (You must hit BREAK to exit this function.)

Other options used in conjunction with the above **uulog** options:

−x	Looks in the **uuxqt** log file for the given system.
−*number*	Indicates that a **tail** command of *number* lines should be executed.

uuname options

The following options are used by **uuname** *only:*

−c	Returns the names of systems known to **cu**(1). This list should be the same as the list of systems known to **uucp**, unless your machine is using different *Systems* files for **cu** and **uucp**. See the *Sysfiles* file.
−l	Return the local system name.

NOTES

The domain of remotely accessible files may (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by pathname. Ask a responsible person on the remote system to send them to you. For the same reasons, you will probably not be able to send files to arbitrary pathnames. As distributed, the remotely accessible files are those whose names begin with **/usr/spool/uucppublic** (equivalent to ˜/).

The forwarding of files through other systems may not be compatible with the previous version of **uucp**. If forwarding is used, all systems in the route must have the same version of **uucp**.

BUGS

Protected files and files in protected directories that are owned by the requestor can be sent by **uucp**. However, if the requestor is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

FILES

/usr/spool/uucp	Spool directory
/usr/spool/uucppublic	Public directory for receiving and sending (PUBDIR)
/usr/lib/uucp/*	Other data and program files

SEE ALSO

mail (1), uustat (1C), uux (1C), uuxqt (1M), chmod (2).

NAME

uuencode,uudecode – encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [*source*] *remotedest* | **mail** *sys1!sys2!..!decode*
uudecode [*file*]

DESCRIPTION

uuencode and **uudecode** are used to send a binary file via **uucp**(1C) or other methods of sending mail. This combination can be used over indirect mail links even when **uusend**(1C) is not available.

uuencode takes the named source file (the default is standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the *remotedest* for re-creation on the remote system.

uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user *decode* should be filtered through the **uudecode** program. This way the file is created automatically without human intervention. This is possible on the uucp network by either using **sendmail**(8) or by making **rmail**(1) be a link to **/usr/ucb/mail** instead of **/bin/mail**. In each case, an alias must be created in a master file to get the automatic invocation of **uudecode**.

If these facilities are not available, the file can be sent to a user on the remote machine who can **uudecode** it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

EXAMPLE

The *remotedest* is the pathname of the file to create on the remote system, for example,

```
$ uuencode /kate/bin/prog1 /tmp/kate.prog1.
```

BUGS

The file is expanded by 35% (three bytes become four plus control information), increasing the transmission time.

The user on the remote system who is invoking **uudecode** (often **uucp**) must have write permission on the specified file.

SEE ALSO

binmail(1), mail(1), uusend(1C), uucp(1C), uux(1C), uuencode(5);
Managing SysV System Software.

NAME

uuencode,uudecode – encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [*source*] *remotedest* | **mail** *sys1!sys2!..!decode*
uudecode [*file*]

DESCRIPTION

uuencode and **uudecode** are used to send a binary file via **uucp**(1C) or other methods of sending mail. This combination can be used over indirect mail links even when **uusend**(1C) is not available.

uuencode takes the named source file (the default is standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the *remotedest* for re-creation on the remote system.

uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user *decode* should be filtered through the **uudecode** program. This way the file is created automatically without human intervention. This is possible on the uucp network by either using **sendmail**(8) or by making **rmail**(1) be a link to **/usr/ucb/mail** instead of **/bin/mail**. In each case, an alias must be created in a master file to get the automatic invocation of **uudecode**.

If these facilities are not available, the file can be sent to a user on the remote machine who can **uudecode** it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

EXAMPLE

The *remotedest* is the pathname of the file to create on the remote system, for example,

```
$ uuencode /kate/bin/prog1 /tmp/kate.prog1.
```

BUGS

The file is expanded by 35% (three bytes become four plus control information), increasing the transmission time.

The user on the remote system who is invoking **uudecode** (often **uucp**) must have write permission on the specified file.

SEE ALSO

binmail(1), mail(1), uusend(1C), uucp(1C), uux(1C), uuencode(5);
Managing SysV System Software.

NAME

uucp, uulog, uuname – UNIX-to-UNIX system copy

SYNOPSIS

uucp [*options*] *source-files destination-file*

uulog [*options*] –s*system*
uulog [*options*] *system*
uulog [*options*] –f*system*

uuname [**–l**] [**–c**]

DESCRIPTION

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A filename may be a pathname on a machine, or may have the following form:

system name!*pathname*

where *system name* is taken from a list of system names that **uucp** knows about. The *system name* may also be a list of names such as:

system name!*system name*!...!*system name*!*pathname*

in which case an attempt is made to send the file via the specified route, to the destination. See NOTES and BUGS below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to forward information.

The question mark (?), asterisk (*), and bracketed ellipsis ([...]) Shell metacharacters appearing in *pathname* are expanded on the appropriate system.

Pathnames may be one of the following (anything else is prefixed by the current directory): A full pathname A pathname preceded by ˜*user* where *user* is a log-in name on the specified system and is replaced by that user's log-in directory A pathname preceded by ˜/*destination* where *destination* is appended to **/usr/spool/uucppublic**. This destination will be treated as a filename unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a slash mark (/). For example, ˜**/dan/** as the destination will make the directory **/usr/spool/uucppublic/dan** if it does not exist and put the requested file(s) in that directory). <.PP If the result is an erroneous pathname for the remote system, the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions. See **chmod(2)** for more information about permissions. All files received by **uucp** will be owned by **uucp**.

uulog queries a summary log of **uucp** or **uuxqt** transactions in the files **/usr/spool/uucp/.Log/uucico/system**, or **/usr/spool/uucp/.Log/uuxqt/system.**

uuname lists the names of systems known to **uucp.**

The DOMAIN/IX version of **uucp** supports the Vadic Autodialer.

OPTIONS

uucp options

The following options are interpreted by **uucp** *only* :

–c	Does not copy the local file to the spool directory for transfer to the remote machine (default).
–C	Forces the copy of local files to the spool directory for transfer.
–d	Makes all necessary directories for the file copy (default).
–f	Does not make intermediate directories for the file copy.
–g*grade*	*Grade* is a single letter/number; lower ascii sequence characters cause the job to be transmitted earlier during a particular conversation.
–j	Output the job identification ASCII string on the standard output. This job identification can be used by **uustat** to obtain the status or terminate a job.
–m	Send mail to the requester when the copy is completed. The **–m** option only works sending files or receiving a single file. Receiving multiple files specified by special shell characters ? * [...] will not activate the **–m** option.
–n*user*	Notifies you on the remote system that a file was sent.
–r	Does not start the file transfer, just queuse the job.
–s*file*	Reports status of the transfer to *file*. Notes that the *file* must be a full path name.
–x*debug_level*	Produces debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information. (Debugging will not be available if **uucp** was compiled with -DSMALL.)

uulog options

The following options are used by **uulog** *only;* they cause **uulog** to print logging information:

–s*sys*	Prints information about file transfer work involving system *sys*.
–f*system*	Does a **tail -f** of the file transfer log for *system*. (You must hit BREAK to exit this function.)

Other options used in conjunction with the above **uulog** options:

–x	Looks in the **uuxqt** log file for the given system.
–*number*	Indicates that a **tail** command of *number* lines should be executed.

uuname options

The following options are used by **uuname** *only:*

–c	Returns the names of systems known to **cu**(1). This list should be the same as the list of systems known to **uucp**, unless your machine is using different *Systems* files for **cu** and **uucp**. See the *Sysfiles* file.
–l	Return the local system name.

NOTES

The domain of remotely accessible files may (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by pathname. Ask a responsible person on the remote system to send them to you. For the same reasons, you will probably not be able to send files to arbitrary pathnames. As distributed, the remotely accessible files are those whose names begin with **/usr/spool/uucppublic** (equivalent to ~/).

The forwarding of files through other systems may not be compatible with the previous version of **uucp**. If forwarding is used, all systems in the route must have the same version of **uucp**.

BUGS

Protected files and files in protected directories that are owned by the requestor can be sent by **uucp**. However, if the requestor is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

FILES

/usr/spool/uucp	Spool directory
/usr/spool/uucppublic	Public directory for receiving and sending (PUBDIR)
/usr/lib/uucp/*	Other data and program files

SEE ALSO

mail (1), uustat (1C), uux (1C), uuxqt (1M), chmod (2).

Also refer to the discussion of **uucp** in *Using Your SysV Environment.*

NAME

uucp, uulog, uuname – UNIX-to-UNIX system copy

SYNOPSIS

uucp [*options*] *source-files destination-file*

uulog [*options*] **–s***system*
uulog [*options*] *system*
uulog [*options*] **–f***system*

uuname [**–l**] [**–c**]

DESCRIPTION

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A filename may be a pathname on a machine, or may have the following form:

system name!pathname

where *system name* is taken from a list of system names that **uucp** knows about. The *system name* may also be a list of names such as:

system name!system name!...!system name!pathname

in which case an attempt is made to send the file via the specified route, to the destination. See NOTES and BUGS below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to forward information.

The question mark (?), asterisk (*), and bracketed ellipsis ([. . .]) Shell metacharacters appearing in *pathname* are expanded on the appropriate system.

Pathnames may be one of the following (anything else is prefixed by the current directory):

- a full pathname
- a pathname preceded by ~*user* where *user* is a log-in name on the specified system and is replaced by that user's log-in directory
- a path name preceded by ~/*destination* where *destination* is appended to **/usr/spool/uucppublic**. This destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a slash mark (/). For example, **~/dan/** as the destination will make the directory **/usr/spool/uucppublic/dan** if it does not exist and put the requested file(s) in that directory).

If the result is an erroneous pathname for the remote system, the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions. See **chmod**(2) for more information about permissions. All files received by **uucp** will be owned by **uucp**.

uulog queries a summary log of **uucp** or **uuxqt** transactions in the files **/usr/spool/uucp/.Log/uucico/system**, or **/usr/spool/uucp/.Log/uuxqt/system**.

uuname lists the names of systems known to **uucp.**

The DOMAIN/IX version of **uucp** supports the Vadic Autodialer.

OPTIONS

The following options are interpreted by **uucp** *only* :

−c	Does not copy the local file to the spool directory for transfer to the remote machine (default).
−C	Forces the copy of local files to the spool directory for transfer.
−d	Makes all necessary directories for the file copy (default).
−f	Does not make intermediate directories for the file copy.
−g*grade*	*Grade* is a single letter/number; lower ascii sequence characters cause the job to be transmitted earlier during a particular conversation.
−j	Output the job identification ASCII string on the standard output. This job identification can be used by **uustat** to obtain the status or terminate a job.
−m	Send mail to the requester when the copy is completed. The **−m** option only works sending files or receiving a single file. Receiving multiple files specified by special shell characters ? * [...] will not activate the **−m** option.
−**n***user*	Notifies you on the remote system that a file was sent.
−r	Does not start the file transfer, just queuse the job.
−s*file*	Reports status of the transfer to *file.* Notes that the *file* must be a full path name.
−x*debug_level*	Produces debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information. (Debugging will not be available if **uucp** was compiled with -DSMALL.)

uulog options

The following options are used by **uulog** *only;* they cause **uulog** to print logging information:

−s*sys*	Prints information about file transfer work involving system *sys.*
−f*system*	Does a **tail -f** of the file transfer log for *system.* (You must hit BREAK to exit this function.)

Other options used in conjunction with the above **uulog** options:

−x	Looks in the *uuxqt* log file for the given system.
−*number*	Indicates that a **tail** command of *number* lines should be executed.

uuname options

The following options are used by **uuname** *only:*

−c	Returns the names of systems known to **cu**. This list should be the same as the list of systems known to **uucp**, unless your machine is using different *Systems* files for *cu* and *uucp*. See the *Sysfiles* file.
−l	Return the local system name.

NOTES

The domain of remotely accessible files may (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by pathname. Ask a responsible person on the remote system to send them to you. For the same reasons, you will probably not be able to send files to arbitrary pathnames. As distributed, the remotely accessible files are those whose names begin with **/usr/spool/uucppublic** (equivalent to ~/).

The forwarding of files through other systems may not be compatible with the previous version of **uucp**. If forwarding is used, all systems in the route must have the same version of **uucp**.

BUGS

Protected files and files in protected directories that are owned by the requestor can be sent by **uucp**. However, if the requestor is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory for receiving and sending (PUBDIR)
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail (1), uustat (1C), uux (1C), uuxqt (1M), chmod (2).

Also refer to the discussion of **uucp** in *Using Your SysV Environment.*

NAME

uuto, uupick – public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [*options*] *source-files destination*
uupick [–s *system*]

DESCRIPTION

uuto sends *source-files* to *destination*. **uuto** uses the **uucp**(1C) facility to send files, while it allows the local system to control the file access. A *source-file* name is a pathname on your machine. Destination has the form

system!*user*

where *system* is taken from a list of system names that **uucp** knows about (see **uuname**(1)); *user* is the log-in name of someone on the specified system.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the **uucp** source. By default this directory is **/usr/spool/uucppublic**. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/*files*.

The destined recipient is notified by **mail**(1) of the arrival of files.

Two **uuto** options are available:

–p	Copy the source file into the spool directory before transmission.
–m	Send mail to the sender when the copy is complete.

uupick accepts or rejects the files transmitted to the user. Specifically, **uupick** searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

```
from system: [file file-name] [dir dirname] ?
```

uupick then reads a line from the standard input to determine the disposition of the file:

<*new-line*>	Go on to the next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir*. If *dir* is not specified as a complete pathname (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
a [*dir*]	The same as **m** except that it moves all the files sent from *system*.
p	Print the content of the file.
q	Stop.

EOT (CTRL/D)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

uupick invoked with the −s*system* option will only search the PUBDIR for files sent from *system*.

WARNINGS

To send files that begin with a dot (for example, **.profile**) the files must by qualified with a dot. For example: **.profile, .prof*, .profil?** are correct; whereas ***prof*, ?profile** are incorrect.

FILES

PUBDIR/usr/spool/uucppublic	Public directory

SEE ALSO

mail(1), uucp(1C), uustat(1C), uux(1C);
uucleanup(1M),
Managing SysV System Software.

NAME

uustat – **uucp** status inquiry and job control

SYNOPSIS

uustat [**–a**]
uustat [**–m**]
uustat [**–p**]
uustat [**–q**]
uustat [–k*jobid*]
uustat [–r*jobid*]
uustat [–s*system*] [–u*user*]

DESCRIPTION

uustat will display the status of, or cancel, previously specified **uucp** commands, or provide general status on **uucp** connections to other systems.

OPTIONS

Only one of the following options can be specified with **uustat** per command execution.

–a	Output all jobs in the queue.
–m	Report the status of accessibility of all machines.
–p	Execute a **ps –flp** for all the process-ids that are in the lock files.
–q	List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in () next to the number of C or X files, it is the age in days of the oldest C./X. file for that system. The ''Retry'' field represents the number of hours until the next possible call. The ''Count'' is the number of failure attempts. NOTE: for systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute.
–k*jobid*	Kill the **uucp** request whose job identification is *jobid*. The killed **uucp** request must belong to the person issuing the **uustat** command unless one is the super-user.
–r*jobid*	Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the deamon.

Either or both of the following options can be specified with **uustat**.

–s*sys*	Report the status of all **uucp** requests for remote system *sys*.
–u*user*	Report the status of all **uucp** requests issued by *user*.

EXAMPLES

The following example shows the output produced by the **–q** option.

```
eagle    3C   04/07-11:07    NO DEVICES AVAILABLE
mh3bs3   2C   07/07-10:42    SUCCESSFUL
```

The above output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system, and are followed by the status of the interaction.

Output for both the **–s** and **–u** options has the following format:

```
eaglen0000     4/07-11:01:03    (POLL)
eagleN1bd7     4/07-11:07       S     eagle dan    522 /usr/dan/A
eagleC1bd8     4/07-11:07       S     eagle dan    59 D.3b2al2ce4924
               4/07-11:07       S     eagle dan    rmail mike
```

With the **–s** and **–u** options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an ``S'' or ``R'' depending on whether the job is to send or request a file. This is followed by the user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (**rmail** – the command used for remote mail), the name of the command. When the size appears in this field, the filename is also given. This can either be the name given by the user or an internal name (e.g., **D.3b2alce4924**) that is created for data files associated with remote executions (**rmail** in this example).

When no options are given, **uustat** outputs the status of all **uucp** requests issued by the current user.

FILES

/usr/spool/uucp/*	Spool directories

SEE ALSO

uucp(1C);
Managing SysV System Software.

NAME

uuto, uupick – public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [*options*] *source-files destination*
uupick [–s *system*]

DESCRIPTION

uuto sends *source-files* to *destination*. **uuto** uses the **uucp**(1C) facility to send files, while it allows the local system to control the file access. A *source-file* name is a path-name on your machine. Destination has the form

system!*user*

where *system* is taken from a list of system names that **uucp** knows about (see **uuname**(1)); *user* is the log-in name of someone on the specified system.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the **uucp** source. By default this directory is **/usr/spool/uucppublic**. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/*files*.

The destined recipient is notified by **mail**(1) of the arrival of files.

Two **uuto** options are available:

–p	Copy the source file into the spool directory before transmission.
–m	Send mail to the sender when the copy is complete.

uupick accepts or rejects the files transmitted to the user. Specifically, **uupick** searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

```
from system: [file file-name] [dir dirname] ?
```

uupick then reads a line from the standard input to determine the disposition of the file:

<*new-line*>	Go on to the next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir*. If *dir* is not specified as a complete pathname (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
a [*dir*]	The same as **m** except that it moves all the files sent from *system*.
p	Print the content of the file.
q	Stop.

EOT (CTRL/D)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

uupick invoked with the –s*system* option will only search the PUBDIR for files sent from *system*.

FILES

PUBDIR/usr/spool/uucppublic   Public directory

WARNINGS

To send files that begin with a dot (for example, **.profile**) the files must by qualified with a dot. For example: **.profile**, **.prof***, **.profil?** are correct; whereas ***prof***, **?profile** are incorrect.

SEE ALSO

mail(1), uucp(1C), uustat(1C), uux(1C);
uucleanup(1M),
Managing SysV System Software.

NAME

uux – UNIX-to-UNIX system command execution

SYNOPSIS

uux [*options*] *command-string*

DESCRIPTION

uux gathers zero or more files from various systems, executes a command on a specified system, and then sends standard output to a file on a specified system.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and filenames may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

Filenames may be one of the following: a full path name a path name preceded by ˜*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory; anything else is prefixed by the current directory. <.PP Any special shell characters such as <>;| should be quoted, either by quoting the entire *command-string*, or by quoting the special characters as individual arguments.

uux attempts to get all files to the execution system. For output files, the filename must be escaped using parentheses. **uux** notifies you if the requested command on the remote system was disallowed. This notification can be turned off by the **–n** option. The response comes by remote mail from the remote machine.

OPTIONS

The following options are interpreted by **uux**:

–	The standard input to **uux** is made the standard input to the *command-string*.
–a*name*	Use *name* as the user identification replacing the initiator user-id. (Notification will be returned to the user.)
–b	Return whatever standard input was provided to the **uux** command if the exit status is non-zero.
–c	Do not copy local file to the spool directory for transfer to the remote machine (default).
–C	Force the copy of local files to the spool directory for transfer.
–g*grade*	*Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.
–j	Output the jobid ASCII string on the standard output which is the job identification. This job identification can be used by **uustat**(1) to obtain the status or terminate a job.
–n	Do not notify the user if the command fails.
–p	Same as –: The standard input to *uux* is made the standard input to the *command-string*.

−r	Do not start the file transfer, just queue the job.
−s*file*	Report status of the transfer in *file*.
−x*debug_level*	Produce debugging output on the standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.
−z	Send success notification to the user.

NOTES

For security reasons, most installations limit the list of commands executable on behalf of an incoming request from **uux**, permitting only the receipt of mail (see **mail**(1)). (Remote execution permissions are defined in **/usr/lib/uucp/Permissions.**)

EXAMPLES

The command

```
uux a!cut −f1 b!/usr/file \(c!/usr/file\)
```

gets /**usr/file** from system **b** and sends it to system **a**, performs a **cut**(1) command on that file and sends the result of the **cut** command to system **c**.

The command

```
uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"
```

will get the **file1** and **file2** files from the machines **usg** and **pwba**, execute a **diff**(1) command, and put the results in **file.diff** in the local **PUBDIR/dan/** directory.

BUGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the **uucp** system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple filename (without path or machine reference) must be unique within the **uux** request. The following command will NOT work:

```
uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

but the command

```
uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

will work. (If **diff**(1) is a permitted command.)

Protected files and files that are in protected directories that are owned by the requestor can be sent in commands using **uux**. However, if the requestor is root, and the directory is not searchable by "other", the request will fail.

FILES

/usr/lib/uucp/spool	Spool directories
/usr/lib/uucp/Permissions	Remote execution permissions
/usr/lib/uucp/*	Other data and programs

SEE ALSO

cut(1), mail(1), uucp(1C), uustat(1C),
Managing SysV System Software.

NAME

val – validate SCCS file

SYNOPSIS

val –

val [–s] [–rSID] [–m*name*] [–y*type*] *files*

DESCRIPTION

val determines if the specified *file* is an SCCS file meeting the characteristics specified by the *option* used.

val has a special argument, –, which reads standard input until it reaches an end-of-file (EOF) condition. Each line read is independently processed as if it were a command line argument list.

val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code on exit as described below.

The 8-bit code returned by **val** is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, –y mismatch;
bit 7 = %M%, **–m** mismatch;

OPTIONS

–s
Silences the diagnostic message normally generated on the standard output for any error detected while processing each named file on a given command line.

–r*SID*
The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. **val** with this *option* c hecks to determine if the *SID* is ambiguous (for example, **r1** is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (for example, **r1.0** or **r1.1.0** are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, **val** with this *option* checks to determine if it actually exists.

–m*name*
Compares the argument value *name* with the SCCS %M% keyword in *file*.

–y*type*
Compares the argument value *type* with the SCCS %Y% keyword in *file*.

NOTE

val can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned – a logical **OR** of the codes generated for each command line and file processed.

DIAGNOSTICS

Use **help**(1) for explanations.

BUGS

val can process up to 50 files on a single command line. Any number above 50 will fail.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccs(1).

NAME

vc – version control

SYNOPSIS

vc [–a] [–t] [–c*char*] [–s] [*keyword=value ... keyword=value*]

DESCRIPTION

vc copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as vc command arguments.

A control statement is a single line beginning with a control character, except as modified by the –t keyletter (see below). The default control character is colon (:), except as modified by the –c keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with ed(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The –a keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter Arguments

–a	Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in vc statements.
–t	All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.
–c*char*	Specifies a control character to be used in place of :.
–s	Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements

:dcl *keyword*[, ..., *keyword*]

Used to declare keywords. All keywords must be declared.

:asg *keyword=value*

Used to assign values to keywords. An asg statement overrides the assignment for the corresponding keyword on the vc command line and all previous asg's for that keyword. Keywords declared, but not assigned values have null values.

:if *condition*

.
.
.

:end

Used to skip lines of the standard input. If the condition is true all lines between the **if** statement and the matching **end** statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening **if** statements and matching **end** statements are recognized solely for the purpose of maintaining the proper **if-end** matching.

The syntax of a condition is:

```
<cond>          ::= [ "not" ] <or>
<or>            ::= <and> | <and> "|" <or>
<and>           ::= <exp> | <exp> "&" <and>
<exp>           ::= "(" <or> ")" | <value> <op> <value>
<op>            ::= "=" | "!=" | "<" | ">"
<value>         ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings
not	may only occur immediately after the *if*, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

```
= != > <     all of equal precedence
&
|
```

Parentheses may be used to alter the order of precedence. Values must be separated from operators or parentheses by at least one blank or tab.

::text
: Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the –a keyletter.

:on

:off
: Turn on or off keyword replacement on all lines.

:ctl char
: Change the control character to char.

:msg message
: Prints the given message on the diagnostic output.

:err message
: Prints the given message followed by:

ERROR: *err statement on line ...* (915)

on the diagnostic output. vc halts execution, and returns an exit code of 1.

EXIT CODES

0 – normal
1 – any error

NAME

vi – screen-oriented (visual) display editor based on **ex**

SYNOPSIS

vi [**–t** *tag*] [**–r** *file*] [**–w***n*] [**–R**] [*+command*] *name ...*
view [**–t** *tag*] [**–r** *file*] [**–w***n*] [**–R**] [*+command*] *name*
vedit [**–t** *tag*] [**–r** *file*] [**–w***n*] [**–R**] [*+command*] *name*

DESCRIPTION

vi (visual) is a display-oriented text editor based on an underlying line editor **ex**(1). It is possible to use the command mode of **ex** from within **vi** and vice-versa.

When using **vi**, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

The *name* argument indicates files to be edited.

The **view** invocation is the same as **vi** except that the **readonly** flag is set.

The **vedit** invocation is intended for beginners. The **report** flag is set to 1, and the **showmode** and **novice** flags are set. These defaults make it easier to get started learning the editor.

OPTIONS

–t *tag*	Edits the file containing the *tag* and positions the editor at its definition.
–r*file*	Recovers *file* after an editor or system crash. If *file* is not specified, a list of all saved files is printed.
–w*n*	Sets the default window size to *n*. This is useful when using the editor over a slow-speed line.
–R	Reads only mode; the **readonly** flag is set, preventing accidental overwriting of the file.
+command	The specified **ex** command is interpreted before editing begins.

VI MODES

Command	Normal and initial mode. Other modes return to command mode upon completion. ESC (escape) is used to cancel a partial command.
Input	Entered by the following options **a i A I o O c C s S R**. Arbitrary text may then be entered. Input mode is normally terminated with ESC character, or abnormally with interrupt.
Last line	Reading input for : / ? or !; terminate with a carriage return to execute, interrupt to cancel.

COMMAND SUMMARY

Sample commands

← ↓ ↑ →	Arrow keys move the cursor
h j k l	Same as arrow keys

i*text*ESC	Insert text *abc*
cw*new*ESC	Change word to *new*
ea*s*ESC	Pluralize word
x	Delete a character
dw	Delete a word
dd	Delete a line
3dd	... 3 lines
u	Undo previous change
ZZ	Exit vi, saving changes
:q!CR	Quit, discarding changes
/*text*CR	Search for *text*
CTRL/U CTRL/D	Scroll up or down
:ex *cmd*CR	Any ex or ed command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways:

line/column number	z G |
scroll amount	CTRL/D CTRL/U
repeat effect	Most of the rest

Interrupting, canceling

ESC	End insert or incomplete cmd
DEL	(Delete or rubout) interrupts
CTRL/L	Reprint screen if DEL scrambles it
CTRL/R	Reprint screen if CTRL/L is → key

File manipulation

:wCR	Write back changes
:qCR	Quit
:q!CR	Quit, discard changes
:e *name*CR	Edit file *name*
:e!CR	Reedit, discard changes
:e + *name*CR	Edit, starting at end
:e +*n*CR	Edit starting at line *n*
:e #CR	Edit alternate file
	Synonym for :e #
:w *name*CR	Write file *name*
:w! *name*CR	Overwrite file *name*
:shCR	Run shell, then return
:!*cmd*CR	Run *cmd*, then return
:nCR	Edit next file in arglist
:n *args*CR	Specify new arglist

CTRL/G	Show current file and line
:ta *tag*CR	To tag file entry *tag*
CTRL/]	:ta, following word is *tag*

In general, any ex or ed command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a carriage return.

Positioning within file

CTRL/F	Forward screen
CTRL/B	Backward screen
CTRL/D	Scroll down half screen
CTRL/U	Scroll up half screen
G	Go to specified line (end default)
/pat	Next line matching *pat*
?pat	Prev line matching *pat*
n	Repeat last / or ?
N	Reverse last / or ?
/pat/+n	nth line after *pat*
?pat?–n	nth line before *pat*
]]	Next section/function
[[	Previous section/function
(	Beginning of sentence
)	End of sentence
{	Beginning of paragraph
}	End of paragraph
%	Find matching ( ) { or }

Adjusting the screen

CTRL/L	Clear and redraw
CTRL/R	Retype, eliminate @ lines
zCR	Redraw, current at window top
z–CR	... at bottom
z.CR	... at center
/pat/z–CR	*pat* line at bottom
zn.CR	Use *n* line window
CTRL/E	Scroll window down 1 line
CTRL/Y	Scroll window up 1 line

Marking and returning

``	Move cursor to previous context
''	... at first non-white in line
m*x*	Mark current position with letter *x*
`*x*	Move cursor to mark *x*
'*x*	... at first non-white in line

Line positioning

H	Top line on screen
L	Last line on screen
M	Middle line on screen
+	Next line, at first non-white
–	Previous line, at first non-white
CR	Return, same as +
↓ or **j**	Next line, same column
↑ or **k**	Previous line, same column

Character positioning

^	First non white
0	Beginning of line
$	End of line
h or →	Forward
l or ←	Backwards
CTRL/H	Same as ←
space	Same as →
f*x*	Find *x* forward
F*x*	f backward
t*x*	Up to *x* forward
T*x*	Back upto *x*
;	Repeat last **f F t** or **T**
,	Inverse of ;
|	To specified column
%	Find matching ({) or }

Words, sentences, paragraphs

w	Word forward
b	Back word
e	End of word
)	To next sentence
}	To next paragraph
(	Back sentence
{	Back paragraph
W	Blank delimited word
B	Back **W**
E	To end of **W**

Corrections during insert

CTRL/H	Erase last character
CTRL/W	Erase last word
erase	Your erase, same as CTRL/H
kill	Your kill, erase input this line
\	Quotes CTRL/H, your erase and kill
ESC	Ends insertion, back to command
DEL	Interrupt, terminates insert
CTRL/D	Backtab over *autoindent*
↑CTRL/D	Kill *autoindent*, save for next
0CTRL/D	... but at margin next also
CTRL/V	Quote non-printing character

Insert and replace

a	Append after cursor
i	Insert before cursor
A	Append at end of line
I	Insert before first non-blank
o	Open line below
O	Open above
r*x*	Replace single char with *x*
R*text***ESC**	Replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since **w** moves over a word, **dw** deletes the word that would be moved over. Double the operator, e.g., **dd** to affect whole lines.

d	Delete
c	Change
y	Yank lines to buffer
<	Left shift
>	Right shift
!	Filter through command
=	Indent for LISP

Miscellaneous Operations

C	Change rest of line (**c$**)
D	Delete rest of line (**d$**)
s	Substitute chars (**cl**)
S	Substitute lines (**cc**)
J	Join lines
x	Delete characters (**dl**)
X	... before cursor (**dh**)
Y	Yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked. However, if a buffer is named, the text in that buffer is put instead.

p	Put back text after cursor
P	Put before cursor
"*x***p**	Put from buffer *x*
"*x***y**	Yank to buffer *x*
"*x***d**	Delete into buffer *x*

Undo, Redo, Retrieve

u	Undo last change
U	Restore current line
.	Repeat last change
"*d* **p**	Retrieve *d*'th last delete

NOTES

In the Domain/OS SysV implementation of **vi**, the w (word) and **CTRL/]** (tag) commands both recognize a $ as part of the word. The w command also recognizes a dash (–) as part of a word, if LISP mode is on.

BUGS

Software tabs using **CTRL/T** work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

FILES

/usr/lib/terminfo/?/* Compiled terminal description database

SEE ALSO

ed(1), edit(1), ex(1).

NAME

vsize – set/display VT100 window settings

SYNOPSIS

vsize [*options*]

DESCRIPTION

The vsize command allows you to set the dimensions of the VT100 emulator window pane. This command is valid only from within the VT100 emulator (which is invoked with the **VT100** command); attempting to use it directly from the shell causes an error.

OPTIONS

If no options are specified, vsize displays the current window pane settings.

–l *n* Specifies the height of the window pane in lines. If this option is omitted, the height remains unchanged.

–c *n* Specifies the width of the window in columns. If this option is omitted, the width remains unchanged.

–std Sets the height of the window to 24 lines and the width to 80 columns. This is the same as saying **–l 24 –c 80**.

EXAMPLES

Invoke VT100 emulator and Display current settings.

```
$ vt100
$ vsize
Screen size is 18 lines by 70 columns.
```

Change the width. Exit the emulator and return to the shell.

```
$ vsize -c 60
Old screen size is 18 lines by 70 columns.
New screen size is 18 lines by 60 columns.
$ *** EOF ***
$
```

NAME

vt100 – VT100 terminal emulator

SYNOPSIS

vt100 [*options*] [*pathname* [*arg1 arg2* ...]]

DESCRIPTION

The **vt100** command creates a window running the VT100 terminal emulator and starts up a shell within the window.

The VT100 terminal emulation package is intended for use with two types of programs. When used in conjunction with remote communications packages such as Domain TCP/IP or X.25, the VT100 terminal emulator allows you to interact with the remote system as if you were logged into a VT100 connected to that system. Using the VT100 terminal emulator with programs that take advantage of VT100 special features allows you to run these programs on a Domain node without having to tailor them to the Domain environment.

pathname [*arg1 arg2* ...] (optional)
Specify the name of a command or program for the shell in the VT100 window to invoke. You must give the full pathname; for example, **/com/ld**. *arg1, arg2*, ... are valid arguments to the selected command (or program): for example, **/com/ld //my_node/my_home_dir**. The default is to invoke the value of the variable $SHELL, or if $SHELL is not set, invoke **/com/sh**.

OPTIONS

If any options are specified, they must precede the argument(s). Once **vt100** is running, you may change the window size with the vsize command.

–std Set up a VT100 window that is 24 lines by 80 columns (the standard size of a VT100 screen).

–lines *n* Set up a VT100 window with the number of lines specified by *n*. The number of lines cannot exceed the number of lines in the DM window running the VT100 emulator.

–columns *n* Set up a VT100 window with the number of columns specified by *n*. The number of columns cannot exceed the number of columns of the DM window running the VT100 emulator.

The VT100 terminal emulation package consists of the following:

- The terminal emulation software, which performs the functions of a VT100 terminal, such as handling VT100-type escape sequences. The terminal emulator redirects the handling of keyboard input and screen output to stream manager operations. The terminal emulator is invoked within a DM window by the **vt100** shell command.

- The terminal emulator driver, which performs keyboard input functions such as erasing or echoing characters.

EXAMPLES

1. Create a window running the VT100 emulator and start a shell running within the window.

   ```
   $ vt100
   ```

2. Open a connection to the remote system specified by *hostname* and create a window running the VT100 emulator.

   ```
   $ vt100 login hostname
   ```

KEYBOARD LAYOUT

The table below shows how the keys on a Domain low-profile keyboard map to the keys of a VT100. This assumes that you are running the VT100 Keyboard Emulation package on your node. Note that the VT100 definitions for the F2, F3, and F7 keys supersede the usual **emt** definitions for these keys.

Domain key	Vt100 keypad
<INS MODE>	<ESC>
<CHAR DEL>	<RUBOUT>
<F2>	<PF1>
<F3>	<PF2>
<F4>	<PF3>
<F5>	<PF4>
SHIFT/<F2>	<7>
SHIFT/<F3>	<8>
SHIFT/<F4>	<9>
SHIFT/<F5>	<->
CTRL/<F2>	<4>
CTRL/<F3>	<5>
CTRL/<F4>	<6>
CTRL/<F5>	<,>
<F6>	<1>
<F7>	<2>
SHIFT/<F6>	<3>
SHIFT/<F7>	<ENTER>
CTRL/<F6>	<0>
CTRL/<F7>	<.>

NAME

wait – await completion of process

SYNOPSIS

wait [*n*]

DESCRIPTION

wait awaits a background process whose process id is *n*, and reports its termination status. If *n* is omitted, all the shell's currently active background processes are waited for, and the return code is zero.

The shell itself executes **wait**, without creating a new process.

CAVEAT

If you get the error message *cannot fork, too many processes*, try using **wait** to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

CAUTIONS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

If *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

SEE ALSO

sh(1).

NAME

wall – write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

wall reads its standard input until an end-of-file. It then sends this message to all currently logged-in users preceded by:

Broadcast Message from ...

It is used to warn all users, typically prior to shutting down the system.

The sender must be super-user to override any protections the users may have invoked (see mesg(1)).

FILES

/dev/tty*

DIAGNOSTICS

Cannot send to ... when the open on a user's tty file fails.

SEE ALSO

mesg(1), write(1)

NAME

wbak – create a magnetic media backup file

SYNOPSIS

wbak –f *fileno* **[–dev | *m*[*unit*] | f | ct]**
[–full|–incr|–af *dtm***|–bef** *dtm***]**
[–fid *id***] [–own** *id***] [–vid** *vol_id***]**
[–sla|–nsla] [–wla|–nwla] [–nhi] [–pdtu]
[–reo] [–reten|–nreten] [–no_eot]
[–sysboot] [–l|–ld|–lf|–ll] [–to *filename***]**
[–type uasc|unstruct|hdru]
[–r] [–stdout] [–presr10] *pathname...*

DESCRIPTION

wbak writes one or more objects to either a removable media, disk file or standard output. These objects may be directory trees, files, or links. For each object, the information saved includes the name, object data, and attributes associated with the object, such as the access control list. This lets you reconstruct files, the directory tree, or any portion of the tree using the **rbak** command.

The **wbak** and **rbak** commands are intended both for disk backup and for interchanging information between separate Apollo installations. Use the **rwmt** command to read and write magnetic media that are used for interchanging information with non–Apollo installations.

pathname (required) Specify the name of the object to be written to backup media. This may be a directory, file, or link. If it is a file, then the file is written as specified. If it is a link, then the link is resolved and the resolution object is written to backup media. If it is a directory, all subordinate files and subdirectories in the tree are written. Note that the pathname specified reflects the way the directory is stored on the backup media, and that the same name must be used when reading files using pathnames in **rbak**. Multiple pathnames and wildcarding are permitted. If you omit this argument, **wbak** will prompt you for it. You may specify a hyphen (–) as an argument to direct **wbak** to standard input for further arguments and options.

OPTIONS

The **–f** option is required, as it specifies where on the backup media the new file is to be written. If you omit it, **wbak** will prompt you for it.

Tape File Identifiers

–fid *file_id* Specify a 1–17 character file ID to be written in the file header label for use when writing a file to a labeled volume. If this option is omitted, the file is not named and can only be restored by the file number.

–f [*position*]	Specify the file position for the write operation. Valid values for *position* are **cur**, **end**, or a nonzero integer. A position of **cur** specifies that the file should be written at the current position on the backup media; the media must have been previously written by **wbak** and its position must not have been disturbed. A position of **end** specifies that the file should be written at the end of the backup media file set. This causes **wbak** to append the specified disk file (*pathname* argument) to the very end of the file set. A position specified by a nonzero integer value causes the file to be written at that absolute position in the backup media volume. If multiple *pathname* arguments are supplied, the value of *position* is incremented by one after each file has been written. The default value for *position* is 1.

Mode Control

The object specified by the *pathname* argument must be a directory for either **–full** or **–incr** to have meaning.

–full (default)	Specify a full backup; save all files in specified trees.
–incr	Specify an incremental backup; save files that were modified since the last backup recorded in the **backup_history** file stored in the *pathname* directory.
–af *dtm*	Save all files modified after the given date and time; *dtm* is in the form *yy/mm/dd.hh:mm*. The date defaults to today, and the time to midnight if either of those are omitted from *dtm*.
–bef *dtm*	Save all files last modified before the given date and time.

Label Control

–wla (default)	Write the backup media volume label if the backup file number is 1.
–nwla	Suppress writing of the backup media volume label.
–own *id*	Specify backup media volume owner (1–14 character name). This option is only meaningful when used with the **–wla** option.
–vid *vol_id*	Specify a 1–6 character volume ID for use when labeling a volume. This option is only meaningful when the backup file number is 1. The default volume ID is ' ' (blank).

–sla (default)	Display the label information written for this backup file on standard output.
–nsla	Suppress output of label information.

Listing Control

You may include the **–l** option, or any combination of **–ld**, **–lf**, and **–ll**.

–l	Write the names of all files, directories, and links saved to standard output.
–lf	Write the names of all files saved to standard output.
–ld	Write the names of all directories saved to standard output.
–ll	Write the names of all links saved to standard output.

Backup Device Control

–dev *d*[*unit*]
: Specify device type and unit number. *d* must be either **m** (for reel–to–reel magnetic tape), **ct** (for cartridge tape), or **f** (for floppy), depending on which drive is being used. *unit* is an integer (0–3). Both are required for reel–to–reel tapes (that is, **–dev m2**). A unit number is not required for floppy disks and cartridge tapes (that is, **–dev f**). If this option is omitted, rbak assumes device **m0**.

  CAUTION: Floppy disk support for this command is limited. In particular, error detection during reads and writes is poor. do not use this command with floppy disks when the data being placed on the floppy disks are critical and unrecoverable.

–to *filename*
: Backup output is written to the specified streams object rather than removable media. This can then be restored using the **–from** option in **rbak**. If the file already exists, use the **–r** option to replace it. If **–type** option is not specified the file will be assigned the default type. You cannot use the **–file** *n* option with streams.

–type [uasc | **unstruct** | **hdru**]
: Specify the type of the object *filename*. It can be one of ASCII (**uasc**), Unstructured (**unstruct**) or Streams header-undefined (**hdru**) type.

–r
: If the object specified with the **–to** option already exists, this option allows it to be replaced. The type of *filename* is however left unchanged.

–stdout	The backup output is written to standard output.
–reo	Force previous backup media volume to be reopened, and suppress reading of backup media volume label. Use only when backup media has not been repositioned since last **wbak** or **rbak**.

Special Cartridge Tape Control Options

–reten	Retension the cartridge tape (unwind to the end, then rewind). This can be helpful if you have encountered cartridge tape reading errors. Retensioning requires about 1.5 minutes to complete.
–nreten (default)	Do not retension the cartridge tape.
–no_eot	Suppress the writing of two tape marks at the end of the tape file, which are the standard signal for end of tape. The cartridge can't position between the two tapemarks to be ready for a successive call to **wbak** (as it does on magtape), without rewinding the tape and searching forward, so this option speeds up multiple invocations of wbak. It should not be used on the last invocation of **wbak**. Also, **–f cur** should be used on all **wbak** invocations in a series except the first one.
–sysboot	Permit use of a bootable tape that has a special boot program at the beginning. This option causes **wbak** to skip over the first file on the tape. This option is only necessary when the first file on the tape is being written (**–f** 1).

Miscellaneous Control Options

–nhi	Suppress update of the backup history file.
– (hyphen)	Read standard input for further arguments or options; input is accepted until **wbak** receives an EOF.
–pdtu	Preserves the last date/time–used information on objects. After each object is backed up on tape, the date/time–used information is reset to the value it had before the backup.
–presr10	Allows you to make a tape with pre-SR10 format from an SR10 node. This tape will have no ACLs by default. You can restore it to a pre-SR10 volume by means of the pre-SR10 **rbak**. If you make a tape without this option it will not be readable on a pre-SR10 system.

EXAMPLES

$ **wbak //mask/wby -f 1 -af 87/11/19.12.00 -fid wby -L**

This command writes the directory **//mask/wby** to tape. The directory is written out to tape file one, and the file ID **wby** is written to the file's label. Disk files from directory **wby** are written to the tape only if they have been modified since noon on November 19, 1987. The label and the names of the files written to tape are printed to standard output.

When this command is executed, **wbak** writes the following information to standard output:

```
Label:
   File number:    1
   File section:   1
   File id:        wby
   Date written:    1987/11/20 10:47:58 EST

Starting write:

(file) "//mask/wby/among" written
(file) "//mask/wby/school" written
(file) "//mask/wby/children" written
(file) "//mask/wby/backup_history" written
(dir)  "//mask/wby/" written.

Write complete.
```

This command backs up the entire contents of the node whose entry directory name is **gooey**. Note that the file ID is specified as **"node 27 backup"** to make it easy to identify when you want to reload it, and that the command assigns volume and owner IDs.

$ **wbak -f 1 -own "john doe" -vid "volbk2" -fid "node 27 backup" //gooey**

When this command is executed, **wbak** writes the following information to standard output:

```
Label:
   Volume id:     VOLBK2
   Owner id:      john doe
   File number:   1
   File section:  1
   File id:       n 27 backup
   File written:  1987/02/17 18:00:39 EST

Starting write:

Write complete.
```

This command uses wildcards to match only those files in the **ug** subdirectory of the current working directory whose names begin with the letters **a** through **f** and end with **_example**.

$ wbak –f 1 –own "john doe" –vid "volbk1" ug/[a–f*]_example –l

When this command is executed, **wbak** writes the following information to standard output:

```
Label:
   Volume id:     VOLBK1
   Owner id:      john doe
   File number:   1
   File section:  1
   File id:       (no id specified)
   File written:  1988/02/17 17:58:52 EST

Starting write:

(file) "ug/cmf_example" written.
(file) "ug/cmt_example" written.
(file) "ug/cpboot_example" written.
(file) "ug/cpf_example" written.
(file) "ug/cpt_example" written.
(file) "ug/fpat_example" written.
(file) "ug/fppmask_example" written.
(file) "ug/fst_example" written.

Write complete.
```

$ wbak src –to /backup/bck_out.file

This command writes the backup output for the directory **src** to the file **/fred/bck_out.file**. The directory can be restored in either of the following two ways :

rbak src –from /backup/bck_out.file
or
cat /fred/bck_out.file | rbak src –stdin

Using streams as a backup output media, it is possible to stage the backup output to intermediate disks and then use **rwmt** to write the intermediate file to the magnetic tape. The sequence to use is as follows

$ wbak //otter –to //backup/ot wbak //otter –to //backup/tmp1

This writes the backup output to an intermediate file //backup/tmp1 followed by

rwmt –f 2 –w //backup/tmp1 –raw –rl 8192 –nobs –ansi

When the magtape unit is available at a later time the intermediate file is written to the magtape. Note that it is ESSENTIAL to use the **–raw**, **–rl** 8192 and the **–nobs** options of **rwmt**, for **rbak** to be able to read the backup from tape. All tapes used for this must must have the ANSI speified volume label. You can only use this sequence for magnetic tapes. **rbak** will not be able to restore data written using the above sequence for cartridge tapes instead of magnetic tapes. This sequence has exactly the same effect as using

wbak //otter –dev mt –f 2

You can then use **rbak** as follows to retrieve the data

rbak //otter –f 2 –dev mt

SEE ALSO

rbak(1), rwmt(1)

NAME

wc – word count

SYNOPSIS

wc [**–lwc**] [*names*]

DESCRIPTION

wc counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or newlines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **–lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

NAME

what – identify SCCS files

SYNOPSIS

what [–s] *files*

DESCRIPTION

whatfl searches the given files for all occurrences of the pattern that get(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ", >, newline, \, or null character. For example, if the C program in file f.c contains

char ident[] = " @(#)identification information ";

and f.c is compiled to yield f.o and a.out, then the command

what f.c f.o a.out

will print

```
f.c:      identification information
f.o:      identification information
a.out:    identification information
```

get(1), which automatically inserts identifying information, can also be used where the information is inserted manually.

OPTIONS

–s	Quit after finding the first occurrence of pattern in each file.

DIAGNOSTICS

Exit status is 0 if any matches are found, otherwise 1. Use **help**(1) for explanations.

BUGS

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

SEE ALSO

get(1).

NAME

who – who is on the system

SYNOPSIS

who [−bdHlpqrstTu] [*file*]

who [−bdHlpqstTu] [**−a** | **−d** | **−n** *arg*]

who am i

who am I

DESCRIPTION

who lists the name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID for each current UNIX system user. It examines the /etc/**utmp** file at login time to obtain its information. If *file* is given, that file (which must be in **utmp**(4) format) is examined. *file* is usually /etc/**wtmp**, which contains a history of all the logins since the file was last created.

who with either the **am i** or **am I** option identifies the invoking user.

The general format for output is:

name [*state*] *line time* [*idle*] [*pid*] [*comment*] [*exit*]

With options, **who** can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the **init** process.

OPTIONS

−a Processes /etc/**utmp** or the named *file* with all options turned on.

−b Indicates the time and date of the last reboot.

−d Displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by **wait**(2)), of the dead process. This can be useful in determining why a process terminated.

−fnodefile Specifies nodes for which /etc/**utmp** are processed. *nodefile* should contain lines in the form: //**nodename** or **[net.]nodeid**, one per line. A pound sign (#) in the first column causes that line to be treated as a comment.

−H Prints column headings above the regular output.

−l Lists only those lines on which the system is waiting for someone to login. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

−**n***arg1* [*,arg2*]
Specificies nodes for which /etc/**utmp** are processed. *arg* lists the nodes to be processed, and should be in the form: //**nodename** or **[net.]nodeid**. If more than one node is specified they should be either separated by commas or separated by whitespace and the entire argument in quotes.

−p Lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in /etc/**inittab**. The *state*, *line*, and *idle* fields have no meaning. The *comment* field shows the *id* field of the line from /etc/**inittab** that spawned this process. See **inittab**(4).

−q Displays only the names and the number of users currently logged on. When this option is used, all other options are ignored.

−r Indicates the current run-level of the **init** process. In addition, it produces the process termination status, process id, and process exit status (see **utmp**(4)) under the *idle*, *pid*, and *comment* headings, respectively.

−s Lists only the *name*, *line*, and *time* fields. This is the default option.

−t Indicates the last change to the system clock (via **date**(1)) by **root**. See su(1).

−T The same as −s except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a − appears if it is not. **root** can write to all lines having a + or a − in the *state* field. If a bad line is encountered, a ? is printed.

−u Lists only those users who are currently logged in. *name* is the user's login name. *line* is the name of the line as found in the directory /**dev**. *time* is the time that the user logged in. *idle* column contains the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore ''current''. If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked **old**. This field is useful when trying to determine whether a person is working at the terminal or not. The *comment* is the comment field associated with this line as found in /etc/**inittab** (see **inittab**(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.

NOTES

All options produce *name*, *line*, and *time* information except −q; only −T produces *state* information.`

After a shutdown to the single-user state, **who** returns a prompt; the reason is that since /etc/**utmp** is updated at login time and there is no login in single-user state, **who** cannot report accurately on this state. **who am i**, however, returns the correct information.

FILES

/etc/**utmp**
/etc/**wtmp**

SEE ALSO

date(1), login(1), mesg(1), su(1M).
init(1M) in the *Managing SysV System Software*.
wait(2), inittab(4), utmp(4) in the *SysV Programmer's Reference*.

NAME

whois – DARPA Internet username directory service

SYNOPSIS

whois *name*

DESCRIPTION

Entering the command **whois help** produces a helpful message similar to the following:

Please enter a name or a handle (''ident'') such as ''Smith'' or ''SRI-NIC''. Starting with a period forces a name-only search; starting with an exclamation point forces handle-only. Examples:

```
        Smith                    [looks for name or handle SMITH  ]
        !SRI-NIC                 [looks for handle SRI-NIC only     ]
        .Smith, John             [looks for name JOHN SMITH only            ]
```

Adding ''...'' to the argument matches anything from that point, e.g. ''ZU...'' matches ZUL, ZUM, etc.

To have the entire membership list of a group or organization, shown with the record, use an asterisk (*) directly preceding the given argument. You can, of course, use an exclamation point and asterisk, or a period and asterisk together.

NAME

write – write to another user

SYNOPSIS

write *user* [*line*]

DESCRIPTION

write copies lines from your terminal to that of another user. When first called, it sends the message:

> **Message from** *yourname* (tty??) [*date*]...

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed **mesg n**. At that point **write** writes **EOT** on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., **tty00**); otherwise, the first writable instance of the user found in **/etc/utmp** is assumed and the following message posted:

> *user* is logged on more than one place.
> You are connected to "*terminal*".
> Other locations are:
> *terminal*

Permission to write may be denied or granted by use of the **mesg**(1) command. Writing to others is normally allowed by default. Certain commands, such as **pr**(1) disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write-inhibited terminal.

If the character **!** is found at the beginning of a line, **write** calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using **write** : when you first write to another user, wait for them to write back before starting to send. Each person should end a message with a distinctive signal (i.e., (o) for "over") so that the other person knows when to reply. The signal (oo) (for "over and out") is suggested when conversation is to be terminated.

FILES

/etc/utmp	To find user
/bin/sh	To execute !

DIAGNOSTICS

user is not logged on
: If the person you are trying to write to is not logged on.

Permission denied
: If the person you are trying to write to denies that permission (with mesg).

Warning: cannot respond, set mesg −y
: If your terminal is set to **mesg n** and the recipient cannot respond to you.

Can no longer write to user
: If the recipient has denied permission (by using **mesg n**) after you had started writing.

SEE ALSO

mail(1), mesg(1), pr(1), sh(1), who(1).

NAME

xargs – construct argument list(s) and execute command

SYNOPSIS

xargs [*flags*] [*command* [*initial-arguments*]]

DESCRIPTION

xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

command, which may be a shell file, is searched for, using one's **$PATH**. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or newlines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **–i** flag). Flags **–i**, **–l**, and **–n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **–l** vs. **–n**), the last flag has precedence.

Flag Values

–l*number* The specified *command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **–x** is forced.

–i*replstr* Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **–x** is also forced. { } is assumed for *replstr* if not specified.

−n*number*	Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option −x is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.
−t	Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.
−p	Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**−t**) is turned on to print the command instance to be executed, followed by a ?... prompt. A reply of y (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.
−x	Causes **xargs** to terminate if any argument list would be greater than *size* characters; −x is forced by the options **−i** and **−l**. When neither of the options **−i**, **−l**, or **−n** are coded, the total length of all arguments must be within the *size* limit.
−s*size*	The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If −s is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.
−e*eofstr*	The specified *eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical **EOF** string if −e is not coded. The value −e with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). **xargs** reads standard input until either end-of-file or the logical **EOF** string is encountered.

xargs will terminate if either it receives a return code of **−1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly exit (see **sh**(1)) with an appropriate value to avoid accidentally returning with **−1**.

EXAMPLES

The following will move all files from directory **$1** to directory $2, and echo each move command just before doing it:

```
        ls $1 | xargs -i -t
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file **log**:

```
(logname; date; echo $0 $*) |
```

The user is asked which files in the current directory are to be archived and archives them into **arch** one at a time,

```
ls | xargs -p -l ar
```

or many at a time:

```
ls | xargs -p -l |
```

The following will execute **diff**(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

SEE ALSO

sh(1).

NAME

xdmc – execute a DM command from the shell

SYNOPSIS

xdmc *dm_command* [*args*...]

DESCRIPTION

xdmc allows you to invoke Display Manager commands from the command shell or from within a shell script. This is similar to pressing <CMD> on the keyboard and then typing the DM command in the DM input window, which is the usual way to perform DM operations.

dm_command (required)	Specifies the Display Manager command to be executed.
args ... (optional)	Specifies any arguments to be passed to the DM command. These are sent directly to the DM without further processing by the command shell. Default if omitted: no arguments passed

EXAMPLES

```
$ xdmc dq
```

Cause the DM to send a quit fault to the current process.

```
$ xdmc cp /com/sh
```

Cause the DM to create a new process and invoke the shell. This is the same as pressing <SHELL>.

NAME

yacc – yet another compiler-compiler

SYNOPSIS

yacc [**–vdlt**] *grammar*

DESCRIPTION

The **yacc** command converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; **lex**(1) is useful for creating lexical analyzers usable by **yacc**

If the **–v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **–d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the **yacc** -assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

If the **–l** flag is given, the code produced in **y.tab.c** will *not* contain any **#line** constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, when **yacc**'s **–t** option is used, this debugging code will be compiled by default. Independent of whether the **–t** option was used, the runtime debugging code is under the control of **YYDEBUG**, a preprocessor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

CAVEAT

Because filenames are fixed, at most one **yacc** process can be active in a given directory at a given time.

FILES

y.output	
y.tab.c	
y.tab.h	Defines for token names
yacc.tmp	
yacc.debug, yacc.acts	Temporary files

usr/lib/yaccpar
Parser prototype for C programs

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

SEE ALSO

lex(1).

NAME

intro – introduction to games

DESCRIPTION

This section describes the recreational and educational programs found in the directory **/usr/games**.

SEE ALSO

domain(6)

NAME

domain – Domain/OS-specific games

DESCRIPTION

While providing all of the significant functionality of System V Release 3, Domain/OS SysV actually represents only a subset of the greater functionality of Domain/OS. Furthermore, Domain/OS SysV omits some features of System V Release 3, that are irrelevant to Apollo® workstations. The following paragraphs list additional games available in the Domain/OS /usr/games directory.

Domain/OS Additions to the SysV Environment

The /usr/games directory includes standard 4.3BSD games, System V games, and Domain/OS-specific games Pages that describe Domain/OS-specific games have the heading, "Domain/OS SysV"; pages documenting standard UNIX games are identified with the heading "SysV".

bgcolor	Make interesting background colors
bj	The game of blackjack
btlfortune	Bell Telephone Labs' version of fortune
btlgammon	Bell Telephone Labs' version of fortune
btlhangman	Bell Telephone Labs' version of hangman
craps	The game of craps
dmoire	Domain/Dialogue-based moire generator
factor	Factoring program
flake	Induce terminal dandruff
mastermind	Mastermind guessing game
maze	Generate a maze
melt	"Melt" the screen
moo	Guessing game
primes	Print prime numbers
puzzle	A puzzle game
random	Random number generator
revscr	Reverse screen
scramble	Turn your screen into a scramble puzzle
teachgammon	Teach the game of backgammon
ttt	The game of tic-tac-toe
vine	Grow vines

SEE ALSO

domain(1), intro(6), domain(1M)

NAME

arithmetic – provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [+–x/] [*range*]

DESCRIPTION

arithmetic presents simple arithmetic problems, and waits for you to type an answer. If the answer is correct, it replies "Right!", and supplies a new problem. If the answer is wrong, it replies "What?", until you respond correctly.

The first optional argument determines the kind of problem to be generated. A plus sign (+), minus sign (–), lowercase x, and a slash (/) produce addition, subtraction, multiplication, and division problems respectively. Specifying more than one of these characters on a command line generates a variety of problem types, all mixed in random order. Specifying any characters other than the four mentioned here also produces a random mix of problem types. If you specify no argument to **arithmetic**, subtraction problems appear by default.

The second optional argument is *range*, a decimal number. If used, all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to this number. The default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

Every twenty problems, it publishes statistics on correctness and the time required to answer. Specifically, the program tells you the number of correct and incorrect answers that you have given, as well as the total percentage of those correct. It also tells you how much time (in seconds) has elapsed, and the average number of seconds it took you to answer each problem. For example, the program may output something like this:

```
Rights 20; Wrongs 1; Score 95%
Total time 50 seconds; 2.5 seconds per problem
```

To quit the program, type an interrupt (usually CTRL/C).

NOTES

As a matter of educational philosophy, **arithmetic** does not supply correct answers, since the learner should be able to calculate them. Thus, it does not try to teach number facts, but instead serves as a drill program for those just past the first learning stage of arithmetic. Usually, the most relevant statistic it provides is time per problem, not percent correct.

NAME

backgammon – the game of backgammon

SYNOPSIS

/usr/games/backgammon [*options*] [*file*]

DESCRIPTION

This program lets you play backgammon against the computer or against a friend. All commands consist of only one letter, so you don't need to type a carriage return except at the end of a move. The program is mostly self documenting; typing a question mark (?) will usually get some help. If you answer y when the program asks if you want the rules, you will get text explaining the rules of the game, some hints on strategy, instructions on how to use the program, and a tutorial consisting of a practice game against the computer.

OPTIONS

–n	Don't ask for rules or instructions.
–r	Player is red (implies **n**).
–w	Player is white (implies **n**).
–b	Two players, red and white (implies **n**).
–pr	Print the board before red's turn.
–pw	Print the board before white's turn.
–pb	Print the board before both players' turns.

Several arguments may be concatenated together. If your terminal has capabilities for direct cursor movement, **backgammon** ''fixes'' the board after each move so that you need not reprint the board each time. (In this case, all **–pr**, **–pw**, and **–pb** options are ignored.)

COMMANDS

When the program prompts by typing only your color, type a space or carriage return to roll, or

d	Double
p	Print the board
q	Quit
s	Save the game for later

When the program prompts with ''Move:'', type

p	Print the board
q	Quit
s	Save the game

or a move, which is a sequence of

s-f	Move from s to **f**
s/**r**	Move one man on s the roll **r**

separated by commas or spaces and ending with a newline. Available abbreviations are

s-f1-f2	means **s-f1,f1-f2**
s/**r1r2**	means s/**r1**,s/**r2**

Use **b** for bar and **h** for home, or 0 or 25 as appropriate.

FILES

/etc/termcap	Terminal capability database
/usr/games/teachgammon	Rules and tutorial

NAME

banner – print large banner on printer

SYNOPSIS

/usr/games/banner [–w*n*] [*message*]

DESCRIPTION

banner prints a large, high quality banner on the standard output. If *message* is omitted, **banner** prompts for and reads one line of its standard input. The output should be printed on a hardcopy device up to 132 columns wide, with no breaks between the pages.

OPTIONS

–w*n* Size output for device of width *n*. If *n* is omitted, a value of 80 is assumed.

BUGS

Several ASCII characters are not defined, notably <, >, [,], \, ^, _, {, }, |, and ~. Also, the characters ", ', and & are funny looking (but in a useful way.)

The –w option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

NAME

battlestar – a tropical adventure game

SYNOPSIS

battlestar [−r]

DESCRIPTION

battlestar is an adventure game in the classic style. However, it's slightly less of a puzzle and more a game of exploration. There are a few magical words in the game, but on the whole, simple English should suffice to make one's desires understandable to the parser.

THE SETTING

In the days before the darkness came, when battlestars ruled the heavens...

Three He made and gave them to His daughters,
Beautiful nymphs, the goddesses of the waters.
One to bring good luck and simple feats of wonder,
Two to wash the lands and churn the waves asunder,
Three to rule the world and purge the skies with thunder.

In those times great wizards were known and their powers were beyond belief. They could take any object from thin air, and, uttering the word 'su' could disappear.

In those times men were known for their lust of gold and desire to wear fine weapons. Swords and coats of mail were fashioned that could withstand a laser blast.

But when the darkness fell, the rightful reigns were toppled. Swords and helms and heads of state went rolling across the grass. The entire fleet of battlestars was reduced to a single ship.

COMMANDS

For commands which manipulate objects, the "shadow" of the next word stays around if you want to take advantage of it: that is, saying **take knife** and then **drop** will drop the knife you just took.

take	Take an object
drop	Drop an object
wear	Wear an object you are holding
draw	Carry an object you are wearing
puton	Take an object and wear it
take off	Draw an object and drop it

throw *object direction*
Throw an object in the specified direction

N S E W	Move in one of the four compass directions. You may use these commands only if you have a compass.
R L A B	Move right/left/ahead/back. Directions printed in room descriptions are always printed using these relative directions.
inven	Display inventory
save	Save the game in a file named **Bstar**. Saved games can be restarted using the **–r** option.
!	Escape to a shell
score	Display current score

BUGS

Countless.

NAME

bcd – convert to antique media

SYNOPSIS

/usr/games/bcd *text*

DESCRIPTION

bcd converts the literal *text* into a form familiar to old-timers.

SEE ALSO

dd(1)

NAME

bgcolor – make interesting background colors

SYNOPSIS

/usr/games/**bgcolor** [*options*]

DESCRIPTION

bgcolor changes the display's background colors.

OPTIONS

–s[lot] *num*	Specify the color as value from the color table. The default value is 4.
–c[olor] *rgbval*	Specify the color as an RGB value.
–f[ade]	Change color continuously, cycling through RGB values.
–i[ncrement] *num*	Change RGB value by *num* for each color update induced by the **–fade** option. Higher values cause the background color to cycle more quickly. Default value is 1.
–d[evice] *name*	Set the device to be used by **bgcolor**. Possible values for *name* are **borrow** (use entire display), **borrow_nc** (same as **borrow** but doesn't clear screen first), **direct** (in current window only), and **bg** (use display background).
–nb	Display no borders around target window.

BUGS

bgcolor produces unpredictable results on monochrome displays.

NAME

bj – the game of blackjack

SYNOPSIS

/usr/games/bj

DESCRIPTION

bj is a serious attempt at simulating the dealer in the game of blackjack (or twenty-one) as might be found in Reno. The following rules apply:

> The bet is $2 every hand.
>
> A player "natural" (black jack) pays $3. A dealer natural loses $2. Simultaneous dealer and player naturals is a "push" (no money exchange).
>
> If the dealer has an ace up, you can make an "insurance" bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where you win $2 if the dealer has a natural, and lose $1 if the dealer does not.
>
> If dealt two cards of the same value, you can "double", that is, play two hands, each with one of these cards. The bet also doubles ($2 on each hand).
>
> If a dealt hand totals 10 or 11, you may "double down". This means that you may double the bet ($2 to $4) and receive exactly one more card on that hand.
>
> Under normal play, you may "hit" (draw a card) as long as your total isn't over twenty-one. If you "bust" (go over twenty-one), the dealer wins the bet.
>
> When you "stand" (decide not to hit), the dealer hits until attaining a total of seventeen or more. If the dealer busts, you win the bet.
>
> If both you and the dealer stand, the one with the largest total wins. A tie is a push.

The machine deals and keeps score. The following questions are asked at appropriate times. You must answer each question by a y and a carriage return for "yes", or just a carriage return for "no".

?	(This means "do you want a hit?")
Insurance?	
Double down?	

Every time the deck is shuffled, the dealer so states and the "action" (total bet) and "standing" (total won or lost) is printed. To exit, type an interrupt and the action and standing are printed.

NAME

boggle – play the game of boggle

SYNOPSIS

/usr/games/boggle [+[+]]

DESCRIPTION

This program is intended for people wishing to sharpen their skills at Boggle (TM Parker Bros.). If you invoke the program with 4 arguments of 4 letters each (*e.g.*, "**boggle appl epie moth erhd**"), the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If you invoke the program without arguments, it will generate a board for you, let you enter words for 3 minutes, and then tell you how well you did relative to **/usr/dict/words**.

The object of Boggle is to find, within 3 minutes, as many words as possible in a 4 by 4 grid of letters. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found..

In interactive play, enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting an interrupt key. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Advanced players may wish to invoke the program with one or two plus signs (+) as the argument. The first + removes the restriction that positions can only be used once in each word. The second + causes a position to be considered adjacent to itself as well as its (up to) 8 neighbors.

FILES

/usr/dict/words

NAME

btlfortune – print a random comment

SYNOPSIS

/usr/games/btlfortune

DESCRIPTION

This is Bell Telephone Labs' version of **fortune**. **btlfortune** prints a fortune, anecdote, saying, or other random comment. All lines are derived from the default fortune database in **/usr/games/lib/btlfortunes**.

NAME

btlgammon – the game of backgammon

SYNOPSIS

/usr/games/btlgammon

DESCRIPTION

This is Bell Telephone Labs' version of **backgammon**. It will ask whether you need instructions.

NAME

btlhangman – guess the word

SYNOPSIS

/usr/games/btlhangman [*arg*]

DESCRIPTION

This is Bell Telephone Labs' version of **hangman**. **btlhangman** chooses a word at least seven letters long from a dictionary. You must guess letters, one at a time, until you guess the word.

The optional argument *arg* names an alternate dictionary.

FILES

/usr/games/lib/w2006	Dictionary

NOTES

Hyphenated compounds are run together.

NAME

canfield, cfscores – the solitaire card game canfield

SYNOPSIS

/usr/games/canfield
/usr/games/cfscores [**–a**] [*user*]

DESCRIPTION

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In Canfield, tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types **ht** for his move. Foundation base cards are also automatically moved to the foundation when they become available.

The command **c** causes **canfield** to maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase one's chances of winning.

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as possible without moving any cards from your hand to the talon. (The initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute.

With no arguments, the program **cfscores** prints out the current status of your canfield account. If a user name is specified, it prints out the status of their canfield account. If the **–a** flag is specified, it prints out the canfield accounts for all users that have played the game since the database was set up.

FILES

/usr/games/canfield	The game itself
usr/games/cfscores	The database printer **/usr/games/lib/cfscores** The database of scores

NAME

craps – the game of craps

SYNOPSIS

/usr/games/craps

DESCRIPTION

craps is a form of the game of craps that is played in Las Vegas. The program simulates the *roller*, while you place bets. At any time, you may choose to bet with the roller or with the *House*. A bet of a negative amount is taken as a bet with the House; any other bet is a bet with the roller.

At the start of the game, you have a ''bankroll'' of $2,000. The program begins prompting with:

''bet?''

The bet can be all or part of your bankroll. Any bet over the total bankroll is rejected and the program continues prompting until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (you win or lose, depending on whether the bet is placed with the roller or with the House; the odds are even). The first roll is the roll immediately following a bet:

1. On the first roll:

   7 or 11 -- wins for the roller;

   2, 3, or 12 -- wins for the House;

   any other number is the point, so roll again (Rule 2 applies)

2. On subsequent rolls:

   point -- roller wins;

   7 -- House wins;

   any other number -- roll again.

If you lose your entire bankroll, the House offers to lend you an additional $2,000. The program prompts as follows:

''marker?''

A yes (or y) consummates the loan. Any other reply terminates the game.

If you owe the House money, the House reminds you, before you can place a bet, how many markers are outstanding.

If, at any time, you have outstanding markers and your bankroll exceeds $2,000, the House asks:

''Repay marker?''

A reply of yes (or y) indicates your willingness to repay the loan. If only 1 marker is outstanding, the debt is immediately repaid. However, if more than 1 marker is outstanding, the House asks:

> ''How many?''

markers you want to repay. If you enter an invalid number or just a carriage return, the program prints an appropriate message and prompts with

> ''How many?''

until you provide a valid number.

If you accumulate 10 markers (a total of $20,000 borrowed from the House), the program tells you and then exits.

Should your bankroll exceed $50,000, the House automatically deducts from it the total amount of money needed to pay off all outstanding markers.

If you accumulate $100,000 or more, you break the bank. The program then prompts:

> ''New game?''

to give the House a chance to win back its money.

The program usually considers any reply other than a yes to be a no. Exceptions to this are when the program asks you if you want to place a bet (i.e., bet?) and when it asks how many markers you want to pay off (i.e., How many?).

To exit, send an interrupt (CTRL/I). Before exiting, the program tells you whether you won, lost, or broke even.

MISCELLANEOUS

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.

NAME

cribbage – the card game cribbage

SYNOPSIS

/usr/games/cribbage [**–req**] *name* ...

DESCRIPTION

cribbage allows you to play the card game cribbage. The program plays one hand and you play the other. At the beginning of the game, the program asks if you need to see the rules of the game. If so, it will print out the appropriate section from *According to Hoyle* with **more**(1).

cribbage first asks you whether you wish to play a short game (''once around'', to 61) or a long game (''twice around'', to 121). A response of 's' results in a short game; any other response results in a long game.

At the start of the first game, the program asks you to cut the deck to determine who gets the first crib. You should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. Whoever cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, the program first prints your hand, whose crib it is, and then asks you to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, the program cuts the deck (if it is your crib) or asks you to cut the deck (if it's the program's crib). In the latter case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the player who doesn't have the crib) leading the first card. Play continues until all cards are exhausted. The program keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. The program asks you to score your hand (and the crib, if yours) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck, this is equivalent to choosing the top card.

Cards are specified as rank followed by suit. You may specify ranks by typing a one-character identifier, or by spelling out the rank as a word. Following are valid entries:

a	ace
2	two
3	three
4	four
5	five
6	six
7	seven

8	eight
9	nine
t	ten
j	jack
q	queen
k	king

Suits may be specified as:

s	spaces
h	hearts
d	diamonds
c	clubs

A card may be specified as: <rank> " " <suit> or <rank> " of " <suit>. If the single letter rank and suit designations are used, the space separating the suit and rank may be left out. Also, if only one card of the desired rank is playable, typing the rank is sufficient. For example, if your hand is "2H, 4D, 5C, 6H, JC, KD" and you want to discard the king of diamonds, any of the following could be typed: "k", "king", "kd", "k d", "k of d", "king d", "king of d", "k diamonds", "k of diamonds", "king diamonds", or "king of diamonds".

OPTIONS

–e If you make a mistake scoring your hand or crib, provide an explanation of the correct score. (This is especially useful for beginning players.)

–q Print a shorter form of all messages. (This is only recommended for users who have already played the game without specifying this option.)

–r Instead of asking the player to cut the deck, randomly cut the deck.

NAME

dmoire – Domain/Dialogue-based moire generator

SYNOPSIS

/usr/games/dmoire [**–w** | **–b** | **–i**] [**–inv** **–nb** **–nd**] [**–fg** *color*] [**–bg** *color*]

DESCRIPTION

dmoire creates moire patterns by moving simple geometric shapes across the display and allowing these shapes to overlap. By default, **dmoire** draws in the background of the screen, but will also use a separate window, borrow the display, or even the window its own menus are in.

This program was adapted from a public domain desk accessory.

OPTIONS

–w[indow]	Draw moires in a new window.
–b[orrow]	Borrow the display and draw a simple moire.
–i[ndialog]	Draw inside the menu interface.
–inv[erse]	Invert the background and foreground colors.
–nb[order]	Use with –w to remove the window border.
–nd[ialog]	Do not supply dialog menus.
–fg	Specify a foreground color as an index into your color map (0-255).
–bg	Specify a background color.

BUGS

The **–i** option is a hack which doesn't work very well.

NAME

factor – factoring program

SYNOPSIS

/usr/games/factor [*number*]

DESCRIPTION

factor prints the prime factors of the integer *number* and then exits. If you run **factor** without an argument, it reads lines from the standard input, factoring each number given. Entering a blank line or a non-numeric character causes **factor** to exit.

NAME

fish – play ''Go Fish''

SYNOPSIS

/usr/games/fish

DESCRIPTION

fish plays the game of ''Go Fish,'' a childrens' card game. The object is to accumulate ''books'' of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request.

Eventually, the first player asks for a card which is not in the hand of the second player, who replies ''GO FISH!'' The first player then draws a card from the pool of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing **a**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **j**, **q**, or **k** when asked. Pressing <RETURN> gives you information about the size of your opponent's hand and the pool, and tells you about your opponent's books. Entering the command **p** in place of your first guess puts you into the ''pro'' level. The default is not very difficult.

NAME

flake – induce terminal dandruff

SYNOPSIS

/usr/games/flake

DESCRIPTION

flake causes bits of the screen display to fall off. Type CTRL/Q to put a stop to it.

NAME

fortune – print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune [**–**] [**–wslao**] [**–m** *pattern*] [*file*]

DESCRIPTION

fortune with no arguments prints out a random adage. You may specify an alternate file of adages from which to read. This file must be created by **strfile**(6). Only one such file may be named in a single command line; subsequent ones are ignored.

OPTIONS

–w	Wait before termination for an amount of time calculated from the number of characters in the message. This is useful if **fortune** is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.
–s	Short apothegms only.
–l	Long dicta only.
–o	Choose from an alternate list of aphorisms, often used for potentially offensive ones.
–a	Choose from either list of maxims.
–m *pattern*	Print all fortunes which match the regular expression *pattern*. See regexp(3).

FILES

/usr/games/lib/fortunes.dat

SEE ALSO

regexp(3), strfile(6)

NAME

hangman – Computer version of the game hangman

SYNOPSIS

/usr/games/hangman

DESCRIPTION

In **hangman**, the computer picks a word from the online word list and you must try to guess it. The computer keeps track of which letters have been guessed and how many wrong guesses you have made on the screen in a graphic fashion.

FILES

/usr/dict/words	Online word list

NAME

hunt – a multi-player multi-terminal game

SYNOPSIS

/usr/games/**hunt** [–q] [–m] [*hostname*] [–l *name*]

DESCRIPTION

The object of the game **hunt** is to kill off the other players. There are no rooms, no treasures, and no monsters. Instead, you wander around a maze, find grenades, trip mines, and shoot down walls and players. The more players you kill before you die, the better your score is.

hunt normally looks for an active game on the local network; if none is found, it starts one up on the local host. You may specify the location of the game by providing a *hostname* argument.

hunt only works on screens with at least 24 lines, 80 columns, and cursor addressing. The screen is divided into 3 areas. On the right hand side is the status area. It shows you how much damage you've sustained, how many charges you have left, who's in the game, who's scanning (shown by an asterisk in front of the name), who's cloaked (shown by a plus sign in front of the name), and other players' scores. The rest of the screen is taken up by your map of the maze, except for the 24th line, which is used for longer messages that don't fit in the status area.

The screen symbols used in **hunt** are as follows:

– | +	walls
/\	diagonal (deflecting) walls
#	doors (dispersion walls)
;	small mine
g	large mine
:	shot
o	grenade
O	satchel charge
@	bomb
s	small slime bomb
$	big slime bomb
> < ^ v	you facing right, left, up, or down
} { i !	other players facing right, left, up, or down
*	explosion
\ | /	
– * –	grenade and large mine explosion
/ | \	

Satchel and bomb explosions are larger than grenades (5x5, 7x7, and 3x3 respectively).

Your score is the ratio of number of kills to number of times you entered the game and is only kept for the duration of a single session of **hunt**.

hunt normally drives up the load average to be about (number_of_players + 0.5) greater than it would be without a **hunt** game executing. A limit of three players per host and nine players total is enforced by **hunt**.

COMMANDS

h j k l	Move left/down/up/right. (Note that **hunt** uses the same keys as **vi**(1) for movement.) To change the direction you're facing, use the upper case version of the key.
y u b n	Move in a diagonal direction.
f	Fire in the direction you're facing. Takes 1 charge.
g	Throw grenade in the direction you're facing. Takes 9 charges.
F	Throw satchel charge. Takes 25 charges.
G	Throw bomb. Takes 49 charges.
o	Throw small slime bomb. Takes 15 charges.
O	Throw big slime bomb. Takes 30 charges.
s	Scan (show where other players are). Takes 1 charge.
c	Cloak (hide from scanners). Takes 1 charge.
CTRL/L	Redraw screen
q	Quit

OPTIONS

–l *name*	Specify player name. This command syntax was chosen for **rlogin/rsh** compatibility.
–m	Enter game as monitor: you can see the action but may not participate.
–q	Query the network and report if an active game is found. This is useful for **.login** scripts.

HELPFUL HINTS

- You can only fire in the direction you are facing.
- You can only fire three shots in a row, then the gun must cool.
- A shot only affects the square it hits.
- Shots and grenades move 5 times faster than you do.
- To stab someone, you must face that player and move at them.
- Stabbing does 2 points worth of damage and shooting does 5 points.

- Slime does 5 points of damage each time it hits.
- You start with 15 charges and get 5 more for every new player.
- A grenade affects the nine squares centered about the square it hits.
- A satchel affects the twenty-five squares centered about the square it hits.
- A bomb affects the forty-nine squares centered about the square it hits.
- Slime affects all squares it oozes over (15 or 30 respectively).
- One small mine and one large mine is placed in the maze for every new player. A mine has a 5% probability of tripping when you walk directly at it; 50% when going sideways on to it; 95% when backing up on to it. Tripping a mine costs you 5 points or 10 points respectively. Defusing a mine is worth 1 charge or 9 charges respectively.
- You cannot see behind you.
- Scanning lasts for 20*(number of players) turns. Scanning takes 1 ammo charge, so don't waste all your charges scanning.
- Cloaking lasts for 20 turns.
- Whenever you kill someone, you get 2 more damage capacity points and 2 damage points taken away.
- Maximum type-ahead is 5 characters.
- A shot destroys normal (*i.e.*, non-diagonal, non-door) walls.
- Diagonal walls deflect shots and change orientation.
- Doors disperse shots in random directions (up, down, left, right).
- Diagonal walls and doors cannot be destroyed by direct shots but may be destroyed by an adjacent grenade explosion.
- Slime goes around walls, not through them.
- Walls regenerate, reappearing in the order they were destroyed. One percent of the regenerated walls will be diagonal walls or doors. When a wall is generated directly beneath a player, he is thrown in a random direction for a random period of time. When he lands, he sustains damage (up to 20 percent of the amount of damage he had before impact); that is, the less damage he had, the more nimble he is and therefore less likely to hurt himself on landing.
- The environment variable HUNT is checked to get the player name. If you don't have this variable set, **hunt** will ask you what name you want to play under. If it is set, you may also set up a single character keyboard map, but then you have to enumerate the options; for example,

  setenv HUNT "name=Sneaky,mapkey=zoFfGg1f2g3F4G"

  sets the player name to Sneaky, and the maps z to o, F to f, G to g, 1 to f, 2 to g, 3 to F, and 4 to G. The **mapkey** option must be last.
- It's a boring game if you're the only one playing.

FILES

/usr/games/lib/hunt.driver Game coordinator

NAME

mastermind – Mastermind guessing game

SYNOPSIS

/usr/games/mastermind

DESCRIPTION

This program plays the game of "mastermind". The playing field is a number of slots, in which a number of colored pegs can be placed. The object of the game is to guess what color peg is in each slot. You and the program take turns trying to guess each other's configuration.

Before play begins, **mastermind** asks you whether you want instructions. Respond by typing a "y" for yes or an "n" for no. Following this, you have a chance to decide how many slots and how many colors you want to use. When you enter a guess, type the names of the colors, separated by spaces. When the program makes a guess, respond with two digits separated by spaces.

A guess consists of a possible sequence of colored pegs. The guesser's opponent answers with two numbers: the number of pegs in the guess that exactly match the corresponding pegs in the configuration, and the number of pegs in the guess that match in color but not in position. For example, suppose you are playing with five slots, and the following situation occurs:

```
my configuration:  red    red    yellow   blue   brown
your guess:        blue   red    green    red     red
```

The two numbers would then be 1 and 2. The 1 applies because both you and the program have a red peg in the second slot. In addition, your blue matches the program's blue, though the position is wrong, and one of your reds matches the program's red in the first slot. Only two of your reds match because the program only has two reds in its configuration.

Any time it is your turn to enter a guess, you can ask the program what happened by typing "review" instead of your guess. You get one point for each guess that the program has to make, and it gets one point for each guess that you have to make.

NAME

maze – generate a maze

SYNOPSIS

/usr/games/maze [*seed* [**–n**]]

DESCRIPTION

maze generates a conventional rectangular labyrinth. To set the random number generator to a particular seed value, supply an integer *seed* as the argument. Invoking **maze** with a given seed always produces the same maze.

If you provide a seed value, **maze** also shows the solution to the labyrinth. To suppress this effect, use the **–n** option.

BUGS

Assumes hardcopy output when displaying the solution.

NAME

melt – ''melt'' the screen

SYNOPSIS

/usr/games/melt

DESCRIPTION

This program creates a melted screen effect.

NAME

mille – play Mille Bournes

SYNOPSIS

/usr/games/mille [*file*]

DESCRIPTION

mille plays a two-handed game reminiscent of the Parker Brother's game of Mille Bournes with you. The rules are described below. If a file name is given on the command line, the game saved in that file is started.

When a game is started up, the bottom of the score window will contain a list of commands. They are:

P	Pick a card from the deck. This card is placed in the "P" slot in your hand.
D	Discard a card from your hand. To indicate which card, type the number of the card in the hand (or "P" for the just-picked card) followed by a <RETURN> or <SPACE>. The <RETURN or <SPACE> is required to allow recovery from typos which can be very expensive, like discarding safeties.
U	Use a card. The card is again indicated by its number, followed by a <RETURN> or <SPACE>.
O	Toggle ordering the hand. By default off, if turned on it will sort the cards in your hand appropriately. This is not recommended for the impatient on slow terminals.
Q	Quit the game. This will ask for confirmation, just to be sure. Hitting <DELETE> (or <RUBOUT>) is equivalent.
S	Save the game in a file. If the game was started from a file, you will be given an opportunity to save it on the same file. If you don't wish to, or you did not start from a file, you will be asked for the file name. If you type a <RETURN> without a name, the save will be terminated and the game resumed.
R	Redraw the screen from scratch. The command CTRL/L will also work.
W	Toggle window type. This switches the score window between the startup window (with all the command names) and the end-of-game window. Using the end-of-game window saves time by eliminating the switch at the end of the game to show the final score. Recommended for hackers and other miscreants.

If you make a mistake, an error message will be printed on the last line of the score window, and a bell will beep.

At the end of each hand or game, you will be asked if you wish to play another. If not it will ask you if you want to save the game. If you do, and the save is unsuccessful play will be resumed as if you had said you wanted to play another hand/game. This allows you to use the "S" command to reattempt the save.

CARDS

Here is some useful information. The number in parentheses after the card name is the number of that card in the deck:

Hazard	Repair	Safety
Out of Gas (2)	Gasoline (6)	Extra Tank (1)
Flat Tire (2)	Spare Tire (6)	Puncture Proof (1)
Accident (2)	Repairs (6)	Driving Ace (1)
Stop (4)	Go (14)	Right of Way (1)
Speed Limit (3)	End of Limit (6)	

25 – (10), 50 – (10), 75 – (10), 100 – (12), 200 – (4)

RULES

Object: The point of this game is to get a total of 5000 points in several hands. Each hand is a race to put down exactly 700 miles before your opponent does. Beyond the points gained by putting down milestones, there are several other ways of making points.

Overview: The game is played with a deck of 101 cards. **Distance** cards represent a number of miles traveled. They come in denominations of 25, 50, 75, 100, and 200. When one is played, it adds that many miles to the player's trip so far this hand. **Hazard** cards are used to prevent your opponent from putting down Distance cards. They can only be played if your opponent has a **Go** card on top of the Battle pile. The cards are **Out of Gas**, **Accident**, **Flat Tire**, **Speed Limit**, and **Stop**. **Remedy** cards fix problems caused by Hazard cards played on you by your opponent. The cards are **Gasoline**, **Repairs**, **Spare Tire**, **End of Limit**, and **Go**. **Safety** cards prevent your opponent from putting specific Hazard cards on you in the first place. They are **Extra Tank**, **Driving Ace**, **Puncture Proof**, and **Right of Way**, and there are only one of each in the deck.

Board Layout: The board is split into several areas. From top to bottom, they are: **SAFETY AREA** (unlabeled): This is where the safeties will be placed as they are played. **HAND**: These are the cards in your hand. **BATTLE**: This is the Battle pile. All the Hazard and Remedy Cards are played here, except the **Speed Limit** and **End of Limit** cards. Only the top card is displayed, as it is the only effective one. **SPEED**: The Speed pile. The **Speed Limit** and **End of Limit** cards are played here to control the speed at which the player is allowed to put down miles. **MILEAGE**: Miles are placed here. The total of the numbers shown here is the distance traveled so far.

Play: The first pick alternates between the two players. Each turn usually starts with a pick from the deck. The player then plays a card, or if this is not possible or desirable, discards one. Normally, a play or discard of a single card constitutes a turn. If the card played is a safety, however, the same player takes another turn immediately.

This repeats until one of the players reaches 700 points or the deck runs out. If someone reaches 700, they have the option of going for an *Extension*, which means that the play continues until someone reaches 1000 miles.

Hazard and Remedy Cards: Hazard Cards are played on your opponent's Battle and Speed piles. Remedy Cards are used for undoing the effects of your opponent's nastiness.

> **Go** (Green Light) must be the top card on your Battle pile for you to play any mileage, unless you have played the **Right of Way** card (see below).
> **Stop** is played on your opponent's **Go** card to prevent them from playing mileage until they play a **Go** card.
> **Speed Limit** is played on your opponent's Speed pile. Until they play an **End of Limit** they can only play 25 or 50 mile cards, presuming their **Go** card allows them to do even that.
> **End of Limit** is played on your Speed pile to nullify a **Speed Limit** played by your opponent.
> **Out of Gas** is played on your opponent's *Go* card. They must then play a *Gasoline* card, and then a *Go* card before they can play any more mileage.
> **Flat Tire** is played on your opponent's **Go** card. They must then play a **Spare Tire** card, and then a *Go* card before they can play any more mileage.
> **Accident** is played on your opponent's **Go** card. They must then play a **Repairs** card, and then a **Go** card before they can play any more mileage.

Safety Cards: Safety cards prevent your opponent from playing the corresponding Hazard cards on you for the rest of the hand. It cancels an attack in progress, and *always entitles the player to an extra turn.*

> **Right of Way** prevents your opponent from playing both *Stop* and *Speed Limit* cards on you. It also acts as a permanent *Go* card for the rest of the hand, so you can play mileage as long as there is not a Hazard card on top of your Battle pile. In this case only, your opponent can play Hazard cards directly on a Remedy card other than a Go card.
> **Extra Tank** When played, your opponent cannot play an **Out of Gas** on your Battle Pile.
> **Puncture Proof** When played, your opponent cannot play a **Flat Tire** on your Battle Pile.
> **Driving Ace** When played, your opponent cannot play an **Accident** on your Battle Pile.

Distance Cards: Distance cards are played when you have a **Go** card on your Battle pile, or a Right of Way in your Safety area and are not stopped by a Hazard Card. They can be played in any combination that totals exactly 700 miles, except that *you cannot play more than two 200 mile cards in one hand.* A hand ends whenever one player gets exactly 700 miles or the deck runs out. In that case, play continues until neither someone reaches 700, or neither player can use any cards in their hand. If the trip is completed after the deck runs out, this is called **Delayed Action.**

Coup Fourré: This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In Mille Bournes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, even *before* you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points (see "Scoring" below).

Scoring: Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window have the following meanings:

> **Milestones Played** : Each player scores as many miles as they played before the trip ended.
> **Each Safety** : 100 points for each safety in the Safety area.
> **All 4 Safeties** : 300 points if all four safeties are played.
> **Each Coup Fourré** : 300 points for each Coup Fourré accomplished.

The following bonus scores can apply only to the winning player.

> **Trip Completed** : 400 points bonus for completing the trip to 700 or 1000.
> **Safe Trip** : 300 points bonus for completing the trip without using any 200 mile cards.
> **Delayed Action** : 300 points bonus for finishing after the deck was exhausted.
> **Extension** : 200 points bonus for completing a 1000 mile trip.
> **Shut-Out** : 500 points bonus for completing the trip before your opponent played any mileage cards.

Running totals are also kept for the current score for each player for the hand (**Hand Total**), the game (**Overall Total**), and number of games won (**Games**).

NAME

monop – Monopoly game

SYNOPSIS

/usr/games/monop

DESCRIPTION

monop is reminiscent of the Parker Brothers game Monopoly for 1 to 9 players. The program assumes that the rules of Monopoly are known. It follows the standard rules, with the exception that if a property goes up for auction and there are only two solvent players, no auction is held and the property remains unowned.

The game, in effect, lends the player money, so it is possible to buy something which you cannot afford. However, as soon as a person goes into debt, he must ''fix the problem'', (i.e., make himself solvent) before play can continue. If this is not possible, the player's property reverts to his debtee (either a player or the bank). A player can resign at any time to any person or the bank, which puts the property back on the board unowned.

Any time that the expected response to a question is a string, (for instance, a name, place or person) you can type a question mark (?) to get a list of valid answers. It is not possible to input a negative number, nor is it ever necessary.

COMMANDS

quit Quit the game. Asks for confirmation.

print Print out the current board. The columns have the following meanings:

Name	The first ten characters of the name of the square
Own	The player number of the owner of the property
Price	The cost of the property (if any)
Mg	This field has an asterisk (*) in it if the property is mortgaged
#	If the property is a Utility or Railroad, this is the number of such owned by the owner. If the property is land, this is the number of houses on it.
Rent	Current rent on the property. If it is not owned, there is no rent.

where Tell where all the players are. An asterisk (*) indicates the current player.

own holdings
List your own holdings (money, ''get out of jail free'' cards, and property).

holdings Look at anyone's holdings. The program will ask you whose holdings you wish to look at. When you are finished, type ''done''.

shell Escape to a shell. When the shell dies, the program continues where you left off.

mortgage
Display a list of mortgageable property. Asks which you wish to mortgage.

unmortgage
Unmortgage mortgaged property.

buy Display a list of monopolies on which you can buy houses. If there is more than one, asks you which you want to buy for, and then asks you how many for each piece of property, giving the current amount in parentheses after the property name. If you build in an unbalanced manner (a disparity of more than one house within the same monopoly), asks you to re-enter values.

sell Display a list of monopolies from which you can sell houses. It operates in a manner analogous to **buy**.

card Use a ''get out of jail free'' card. Informs you if you're not in jail, or you don't have such a card.

pay Pay $50 to get out of jail, whence you are put on the Just Visiting space.

trade Trade with another player. Asks you whom you wish to trade with, and then asks you what each wishes to give up. You can get a summary at the end, and, in all cases, it asks for confirmation of the trade before completing it.

resign Resign to another player or the bank. If you resign to the bank, all property reverts to its virgin state, and ''get out of jail free'' cards revert to the deck.

save Save the current game in a file for later play. You can continue play later on either by giving the save file's name as an argument to the **monop** command, or by using the **restore** command (see below). Asks you which file you wish to save to; if the file exists, asks you to confirm that you wish to overwrite it.

restore Read in a previously saved game from a file. Leaves the file intact.

roll Roll the dice and move forward to your new location. If you simply press <RETURN> without entering a command, a **roll** is performed.

FILES

/usr/games/lib/cards.pck	Chance and Community Chest cards

BUGS

No command can be given an argument instead of a response to a query.

NAME

moo – guessing game

SYNOPSIS

/usr/games/moo

DESCRIPTION

moo is a guessing game imported from England. The computer picks a four-digit number which you try to guess; **moo** scores you on each guess. A ''cow'' is a correct digit in an incorrect position. A ''bull'' is a correct digit in a correct position. The game continues until you guess the number (a score of four bulls).

NAME

number – convert Arabic numerals to English

SYNOPSIS

/usr/games/number

DESCRIPTION

number copies the standard input to the standard output, changing each decimal number to a fully spelled-out version.

NAME

primes – print prime numbers

SYNOPSIS

/usr/games/primes [*number*]

DESCRIPTION

primes displays the prime numbers equal to or greater than the input *number*. If you do not provide an argument, **primes** reads a line from the standard input. To exit the program, type an interrupt.

NAME

puzzle – puzzle game

SYNOPSIS

/usr/games/puzzle

DESCRIPTION

puzzle lets you play the familiar 15-tile game. To obtain online help, position the cursor over the puzzle border and press the HELP key.

COMMANDS

Each of the mouse buttons has a particular function:

Left	Use to select items from the menu, and to move individual tiles.
Middle	Back up one move.
Right	Move one step forward in solving the puzzle.

SEE ALSO

scramble(6)

NAME

quiz – test your knowledge

SYNOPSIS

/usr/games/quiz [**–i** *file*] [**–t**] [*category1 category2*]

DESCRIPTION

quiz gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*. If no categories are specified, **quiz** gives instructions and lists the available categories.

quiz tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, **quiz** reports a score and terminates.

The lines of the index file have the following syntax:

```
line     = category newline | category ':' line
category = alternate | category '|' alternate
alternate = empty | alternate primary
primary  = character | '[' category ']' | option
option   = '{' category '}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Use a backslash (\) as with **sh**(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, **quiz** will refrain from asking it.

OPTIONS

–t	Enable tutorial mode, where missed questions are repeated later, and material is gradually introduced as you learn.
–i *file*	Substitute the named file for the default index file.

FILES

/usr/games/quiz.k/*

BUGS

The construct 'a|ab' doesn't work in an information file. Use 'a{b}'.

NAME

rain – animated raindrops display

SYNOPSIS

/usr/games/rain

DESCRIPTION

rain simulates the pitter-patter of raindrops falling on your tty. It looks best at 9600 baud or faster. As with all programs that use **termcap**(3), the TERM environment variable must be set (and exported) to the type of the terminal being used.

rain can only be halted with an interrupt.

FILES

/etc/termcap

NAME

random – random number generator

SYNOPSIS

/usr/games/random [*numbers*] [*characters*]

DESCRIPTION

random is a random number generator that accepts input of numbers or characters. There is about a 50% probability that the program will match and output the lines that you use as input. You can also perform a particular task (e.g., combining lines in a file, listing a directory, etc.) and then use **random** in a pipeline to produce a random sampling of the output. The **random** comand is a likely candidate for use in other games that require the use of a random number generator.

EXAMPLE

The following command line lists a random percentage (about 50%) of the contents of **mydir**:

```
$ ls mydir | random
```

NAME

revscr – reverse screen

SYNOPSIS

/usr/games/revscr

DESCRIPTION

revscr reflects the display image twice, first about the horizontal axis and then about the vertical axis. The effect is to turn the image upside down. Press any key to restore the display to its normal state.

OPTIONS

–short Reflect about the horizontal axis only. Useful on machines which perform the second reflection slowly.

NAME

robots – fight off villainous robots

SYNOPSIS

/usr/games/robots [**–sjta**] [*scorefile*]

DESCRIPTION

robots pits you against evil robots who are trying to kill you (which is why they are evil). Fortunately for you, although they are evil they are not very bright. The robots have a habit of bumping into each other, thus turning themselves into junkpiles. In order to survive, you must get them to kill each other off, since you have no offensive weaponry. You are endowed with one defensive measure: a teleportation device.

When two robots run into each other or a junk pile, they die. If a robot runs into you, you die. When a robot dies, you get 10 points, and when all the robots die, you start on the next field. This keeps up until they finally get you.

Robots are represented on the screen by plus signs (+), the junk heaps resulting from their collisions by asterisks (*), and you (the good guy) by an at sign (@).

Only five scores are allowed per user on the score file. If you make it into the score file, you will be shown the list at the end of the game. If an alternate score file is specified, that will be used instead of the standard file for scores.

COMMANDS

All commands can be preceded by a count. For all commands except w, the program saves you from typos by stopping you short of being eaten. However, with w you take the risk of dying by miscalculation.

h	Move one square left
l	Move one square right
k	Move one square up
j	Move one square down
y	Move one square up and left
u	Move one square up and right
b	Move one square down and left
n	Move one square down and right
. or <SPACE>	Do nothing for one turn
HJKLBNYU	Run as far as possible in the given direction
>	Do nothing for as long as possible
t	Teleport to a random location

w	Wait until they all die (or you do). If you use the w command and survive to the next level, you will get a bonus of 10% for each robot which died after you decided to wait. If you die, however, you get nothing.
q	Quit
^L	Redraw the screen

OPTIONS

–s	Don't play, just show the score file
–j	Jump, *i.e.*, when you run, don't show any intermediate positions; only show things at the end. This is useful on slow terminals.
–t	Teleport automatically when you have no other option. This is a little disconcerting until you get used to it, and then it is very nice.
–a	Advance into the higher levels directly, skipping the lower, easier levels. You receive a 600-point bonus for successful completion of the first field.

FILES

/usr/games/lib/robots_roll	Score file

NAME

sail – multi-user wooden ships and iron men

SYNOPSIS

sail [–s [–l]] [–x] [**–b**] [*num*]

DESCRIPTION

sail is a computer version of Avalon Hill's game of fighting sail originally developed by S. Craig Taylor.

Players of **sail** take command of an old fashioned Man of War and fight other players or the computer. They may re-enact one of the many historical sea battles recorded in the game, or they can choose a fictional battle.

As a sea captain in the **sail** Navy, the player has complete control over the workings of his ship. He must order every maneuver, change the set of his sails, and judge the right moment to let loose the terrible destruction of his broadsides. In addition to fighting the enemy, he must harness the powers of the wind and sea to make them work for him. The outcome of many battles during the age of sail was decided by the ability of one captain to hold the "weather gauge."

OPTIONS

–s Print the names and ships of the top ten sailors.

–l Show the login name. Only effective with –s.

–x Play the first available ship instead of prompting for a choice.

–b No bells.

HISTORICAL INFO

Old Square Riggers were very maneuverable ships capable of intricate sailing. Their only disadvantage was an inability to sail very close to the wind. The design of a wooden ship allowed only for the guns to bear to the left and right sides. A few guns of small aspect (usually 6 or 9 pounders) could point forward, but their effect was small compared to a 68 gun broadside of 24 or 32 pounders. The guns bear approximately like so:

```
   \
    b----------------
---0
    \
     \
      \      up to a range of ten (for round shot)
       \
        \
         \
```

An interesting phenomenon occurred when a broadside was fired down the length of an enemy ship. The shot tended to bounce along the deck and did several times more damage. This phenomenon was called a rake. Because the bows of a ship are very strong and present a smaller target than the stern, a stern rake (firing from the stern to the bow) causes more damage than a bow rake.

```
        b
       00    ----   Stern rake!
        a
```

Most ships were equipped with carronades, which were very large, close range cannons. American ships from the revolution until the War of 1812 were almost entirely armed with carronades. The period of history covered in **sail** runs approximately from the 1770's until the end of Napoleonic France in 1815.

Fighting ships came in several sizes classed by armament. The mainstays of any fleet were its "Ships of the Line", or "Line of Battle Ships". They were so named because these ships fought together in great lines. They were close enough for mutual support, yet every ship could fire both its broadsides. We get the modern words "ocean liner," or "liner," and "battleship" from "ship of the line." The most common size was the the 74 gun two decked ship of the line. The two gun decks usually mounted 18 and 24 pounder guns.

The pride of the fleet were the first rates. These were huge three decked ships of the line mounting 80 to 136 guns. The guns in the three tiers were usually 18, 24, and 32 pounders in that order from top to bottom.

Various other ships came next. They were almost all "razees," or ships of the line with one deck sawed off. They mounted 40-64 guns and were a poor cross between a frigate and a line of battle ship. They neither had the speed of the former nor the firepower of the latter.

Next came the "eyes of the fleet." Frigates came in many sizes mounting anywhere from 32 to 44 guns. They were very handy vessels. They could outsail anything bigger and outshoot anything smaller. Frigates didn't fight in lines of battle as the much bigger 74's did. Instead, they harassed the enemy's rear or captured crippled ships. They were much more useful in missions away from the fleet, such as cutting out expeditions or boat actions. They could hit hard and get away fast.

Lastly, there were the corvettes, sloops, and brigs. These were smaller ships mounting typically fewer than 20 guns. A corvette was only slightly smaller than a frigate, so one might have up to 30 guns. Sloops were used for carrying dispatches or passengers. Brigs were something you built for land-locked lakes.

SAIL PARTICULARS

Ships in **sail** are represented by two characters. One character represents the bow of the ship, and the other represents the stern. Ships have nationalities and numbers. The first ship of a nationality is number 0, the second number 1, etc. Therefore, the first British ship in a game would be printed as "b0". The second Brit would be "b1", and the fifth Don would be "s4".

Ships can set normal sails, called Battle Sails, or bend on extra canvas called Full Sails. A ship under full sail is a beautiful sight indeed, and it can move much faster than a ship under Battle Sails. The only trouble is, with full sails set, there is so much tension on sail and rigging that a well aimed round shot can burst a sail into ribbons where it would only cause a little hole in a loose sail. For this reason, rigging damage is doubled on a ship with full sails set.

A ship with full sails set has a capital letter for its nationality. For example, a French ship, normally "f0", with full sails set would be printed as "F0".

When a ship is battered into a listing hulk, the last man aboard "strikes the colors." This ceremony is the ship's formal surrender. The nationality character of a surrendered ship is printed as "!". Thus, the French ship of the last example would soon be "!0".

A ship has a random chance of catching fire or sinking when it reaches the stage of listing hulk. A sinking ship has a tilde (~) printed for its nationality, and a ship on fire and about to explode has a pound sign (#) printed.

Captured ships become the nationality of the prize crew. Therefore, if an American ship captures a British ship, the British ship will have an "a" printed for its nationality. In addition, the ship number is changed to "&","'", "(", ,")", "*", or "+" depending upon the original number, be it 0,1,2,3,4, or 5. E.g., the "b0" captured by an American becomes the "a&". The "s4" captured by the French becomes the "f*".

MOVEMENT

Movement is the most confusing part of **sail** to many. Ships can head in 8 directions:

```
                               0        0        0
b         b         b0        b         b         b         0b        b
0          0                                                          0
```

The stern of a ship moves when it turns. The bow remains stationary. Ships can always turn, regardless of the wind (unless they are becalmed). All ships drift when they lose headway. If a ship doesn't move forward at all for two turns, it will begin to drift. If a ship has begun to drift, then it must move forward before it turns, if it plans to do more than make a right or left turn, which is always possible.

Movement commands to **sail** are a string of forward moves and turns. An example is "13". It will turn a ship left and then move it ahead 3 spaces. In the drawing above, the "b0" made 7 successive left turns. When **sail** prompts you for a move, it prints three characters of import, e.g.,

    move (7, 4):

The first number is the maximum number of moves you can make, including turns. The second number is the maximum number of turns you can make. Between the numbers is sometimes printed a single quote ('). If the quote is present, it means that your ship has been drifting, and you must move ahead to regain headway before you turn (see note above). Some of the possible moves for the example above are as follows:

```
move (7, 4): 7
move (7, 4): 1
move (7, 4): d              /* drift, or do nothing */
move (7, 4): 6r
move (7, 4): 5r1
move (7, 4): l1r1r2
```

Because square riggers performed so poorly sailing into the wind, if at any point in a movement command you turn into the wind, the movement stops there:

```
move (7, 4): 1114
Movement Error;
Helm: 111
```

Moreover, whenever you make a turn, your movement allowance drops to the lesser of a) what's left and b) what you would have at the new attitude. In short, if you turn closer to the wind, you most likely won't be able to sail the full allowance printed in the "move" prompt.

Old sailing captains had to keep an eye constantly on the wind. Captains in **sail** are no different. A ship's ability to move depends on its attitide to the wind. The best angle possible is to have the wind off your quarter, that is, just off the stern. The direction rose on the side of the screen gives the possible movements for your ship at all positions to the wind. Battle sail speeds are given first, and full sail speeds are given in parenthesis.

```
 0 1(2)
\|/
-^-3(6)
/|\
 | 4(7)
 3(6)
```

Pretend the bow of your ship (the "^") is pointing upward and the wind is blowing from the bottom to the top of the page. The numbers at the bottom "3(6)" will be your speed under battle or full sails in such a situation. If the wind is off your quarter, then you can move "4(7)". If the wind is off your beam, "3(6)". If the wind is off your bow, then you can only move "1(2)". Facing into the wind, you can't move at all. Ships facing into the wind were said to be "in irons".

WINDSPEED AND DIRECTION

The windspeed and direction is displayed as a little weather vane on the side of the screen. The number in the middle of the vane indicates the wind speed, and the + to - indicates the wind direction. The wind blows from the + sign (high pressure) to the - sign (low pressure). E.g.,

```
|
3
+
```

The wind speeds are 0 = becalmed, 1 = light breeze, 2 = moderate breeze, 3 = fresh breeze, 4 = strong breeze, 5 = gale, 6 = full gale, 7 = hurricane. If a hurricane shows up, all ships are destroyed.

GRAPPLING AND FOULING

If two ships collide, they run the risk of becoming tangled together. This is called "fouling." Fouled ships are stuck together, and neither can move. They can unfoul each other if they want to. Boarding parties can only be sent across to ships when the antagonists are either fouled or grappled.

Ships can grapple each other by throwing grapnels into the rigging of the other.

The number of fouls and grapples you have are displayed on the upper right of the screen.

BOARDING

Boarding was a very costly venture in terms of human life. Boarding parties may be formed in **sail** to either board an enemy ship or to defend your own ship against attack. Men organized as Defensive Boarding Parties fight twice as hard to save their ship as men left unorganized.

The boarding strength of a crew depends upon its quality and upon the number of men sent.

CREW QUALITY

The British seaman was world renowned for his sailing abilities. American sailors, however, were actually the best seamen in the world. Because the American Navy offered twice the wages of the Royal Navy, British seamen who liked the sea defected to America by the thousands.

In **sail**, crew quality is quantized into 5 energy levels. "Elite" crews can outshoot and outfight all other sailors. "Crack" crews are next. "Mundane" crews are average, and "Green" and "Mutinous" crews are below average. A good rule of thumb is that "Crack" or "Elite" crews get one extra hit per broadside compared to "Mundane" crews. Don't expect too much from "Green" crews.

BROADSIDES

Your two broadsides may be loaded with four kinds of shot: grape, chain, round, and double. You have guns and carronades in both the port and starboard batteries. Carronades only have a range of two, so you have to get in close to be able to fire them. You have the choice of firing at the hull or rigging of another ship. If the range of the ship is greater than 6, then you may only shoot at the rigging.

The types of shot and their advantages are:

Round	Range of 10. Good for hull or rigging hits.
Double	Range of 1. Extra good for hull or rigging hits. Double takes two turns to load.
Chain	Range of 3. Excellent for tearing down rigging. Cannot damage hull or guns, though.
Grape	Range of 1. Sometimes devastating against enemy crews.

On the side of the screen is displayed some vital information about your ship:

```
Load D! R!
Hull 9
Crew 4 4 2
Guns 4 4
Carr 2 2
Rigg 5 5 5 5
```

"Load" shows what your port (left) and starboard (right) broadsides are loaded with. A "!" after the type of shot indicates that it is an initial broadside. Initial broadside were loaded with care before battle and before the decks ran red with blood. As a consequence, initial broadsides are a little more effective than broadsides loaded later. A "*" after the type of shot indicates that the gun crews are still loading it, and you cannot fire yet. "Hull" shows how much hull you have left. "Crew" shows your three sections of crew. As your crew dies off, your ability to fire decreases. "Guns" and "Carr" show your port and starboard guns. As you lose guns, your ability to fire decreases. "Rigg" shows how much rigging you have on your 3 or 4 masts. As rigging is shot away, you lose mobility.

EFFECTIVENESS OF FIRE

It is very dramatic when a ship fires its thunderous broadsides, but the mere opportunity to fire them does not guarantee any hits. Many factors influence the destructive force of a broadside. First of all, and the chief factor, is distance. It is harder to hit a ship at range ten than it is to hit one sloshing alongside. Next is raking. Raking fire, as mentioned before, can sometimes dismast a ship at range ten. Next, crew size and quality affects the damage done by a broadside. The number of guns firing also bears on the point, so to speak. Lastly, weather affects the accuracy of a broadside. If the seas are high (5 or 6), then the lower gunports of ships of the line can't even be opened to run out the guns. This gives frigates and other flush decked vessels an advantage in a storm. The scenario *Pellew vs. The Droits de L'Homme* takes advantage of this peculiar circumstance.

REPAIRS

Repairs may be made to your Hull, Guns, and Rigging at the slow rate of two points per three turns. The message "Repairs Completed" will be printed if no more repairs can be made.

PECULIARITIES OF COMPUTER SHIPS

Computer ships in **sail** follow all the rules above with a few exceptions. Computer ships never repair damage. If they did, the players could never beat them. They play well enough as it is. As a consolation, the computer ships can fire double shot every turn. That fluke is a good reason to keep your distance. The driver figures out the moves of the computer ships. It computes them with a typical distance function and a depth-first search to find the maximum "score."

COMMANDS

Commands are given to **sail** by typing a single character. You will then be prompted for further input.

f	Fire broadsides if they bear
l	Reload
L	Unload broadsides (to change ammo)
m	Move
i	Print the closest ship
I	Print all ships
F	Find a particular ship or ships (e.g. "a?" for all Americans)
s	Send a message around the fleet
b	Attempt to board an enemy ship
B	Recall boarding parties
c	Change set of sail
r	Repair
u	Attempt to unfoul
g	Grapple/ungrapple

v	Print version number of game
CTRL/L	Redraw screen
Q	Quit
C	Center your ship in the window
U	Move window up
D, N	Move window down
H	Move window left
J	Move window right
S	Toggle window to follow your ship or stay where it is

SCENARIOS

Here is a summary of the scenarios in **sail:**

Ranger vs. Drake:

Wind from the N, blowing a fresh breeze.

(a) Ranger	19 gun Sloop (crack crew) (7 pts)
(b) Drake	17 gun Sloop (crack crew) (6 pts)

The Battle of Flamborough Head:

Wind from the S, blowing a fresh breeze.

This is John Paul Jones' first famous battle. Aboard the Bonhomme Richard, he was able to overcome the Serapis's greater firepower by quickly boarding her.

(a) Bonhomme Rich	42 gun Corvette (crack crew) (11 pts)
(b) Serapis	44 gun Frigate (crack crew) (12 pts)

Arbuthnot and Des Touches:

Wind from the N, blowing a gale.

(b) America	64 gun Ship of the Line (crack crew) (20 pts)
(b) Befford	74 gun Ship of the Line (crack crew) (26 pts)
(b) Adamant	50 gun Ship of the Line (crack crew) (17 pts)
(b) London	98 gun 3 Decker SOL (crack crew) (28 pts)
(b) Royal Oak	74 gun Ship of the Line (crack crew) (26 pts)
(f) Neptune	74 gun Ship of the Line (average crew) (24 pts)
(f) Duc Bougogne	80 gun 3 Decker SOL (average crew) (27 pts)
(f) Conquerant	74 gun Ship of the Line (average crew) (24 pts)
(f) Provence	64 gun Ship of the Line (average crew) (18 pts)
(f) Romulus	44 gun Ship of the Line (average crew) (10 pts)

Suffren and Hughes:
Wind from the S, blowing a fresh breeze.

(b) Monmouth	74 gun Ship of the Line (average crew) (24 pts)
(b) Hero	74 gun Ship of the Line (crack crew) (26 pts)
(b) Isis	50 gun Ship of the Line (crack crew) (17 pts)
(b) Superb	74 gun Ship of the Line (crack crew) (27 pts)
(b) Burford	74 gun Ship of the Line (average crew) (24 pts)
(f) Flamband	50 gun Ship of the Line (average crew) (14 pts)
(f) Annibal	74 gun Ship of the Line (average crew) (24 pts)
(f) Severe	64 gun Ship of the Line (average crew) (18 pts)
(f) Brilliant	80 gun Ship of the Line (crack crew) (31 pts)
(f) Sphinx	80 gun Ship of the Line (average crew) (27 pts)

Nymphe vs. Cleopatre:
Wind from the S, blowing a fresh breeze.

(b) Nymphe	36 gun Frigate (crack crew) (11 pts)
(f) Cleopatre	36 gun Frigate (average crew) (10 pts)

Mars vs. Hercule:
Wind from the S, blowing a fresh breeze.

(b) Mars	74 gun Ship of the Line (crack crew) (26 pts)
(f) Hercule	74 gun Ship of the Line (average crew) (23 pts)

Ambuscade vs. Baionnaise:
Wind from the N, blowing a fresh breeze.

(b) Ambuscade	32 gun Frigate (average crew) (9 pts)
(f) Baionnaise	24 gun Corvette (average crew) (9 pts)

Constellation vs. Insurgent:
Wind from the S, blowing a gale.

(a) Constellation	38 gun Corvette (elite crew) (17 pts)
(f) Insurgent	36 gun Corvette (average crew) (11 pts)

Constellation vs. Vengeance:
Wind from the S, blowing a fresh breeze.

(a) Constellation	38 gun Corvette (elite crew) (17 pts)
(f) Vengeance	40 gun Frigate (average crew) (15 pts)

The Battle of Lissa:
Wind from the S, blowing a fresh breeze.

(b) Amphion	32 gun Frigate (elite crew) (13 pts)
(b) Active	38 gun Frigate (elite crew) (18 pts)

(b) Volage	22 gun Frigate (elite crew) (11 pts)
(b) Cerberus	32 gun Frigate (elite crew) (13 pts)
(f) Favorite	40 gun Frigate (average crew) (15 pts)
(f) Flore	40 gun Frigate (average crew) (15 pts)
(f) Danae	40 gun Frigate (crack crew) (17 pts)
(f) Bellona	32 gun Frigate (green crew) (9 pts)
(f) Corona	40 gun Frigate (green crew) (12 pts)
(f) Carolina	32 gun Frigate (green crew) (7 pts)

Constitution vs. Guerriere:
Wind from the SW, blowing a gale.

(a) Constitution	44 gun Corvette (elite crew) (24 pts)
(b) Guerriere	38 gun Frigate (crack crew) (15 pts)

United States vs. Macedonian:
Wind from the S, blowing a fresh breeze.

(a) United States	44 gun Frigate (elite crew) (24 pts)
(b) Macedonian	38 gun Frigate (crack crew) (16 pts)

Constitution vs. Java:
Wind from the S, blowing a fresh breeze.

(a) Constitution	44 gun Corvette (elite crew) (24 pts)
(b) Java	38 gun Corvette (crack crew) (19 pts)

Chesapeake vs. Shannon:
Wind from the S, blowing a fresh breeze.

(a) Chesapeake	38 gun Frigate (average crew) (14 pts)
(b) Shannon	38 gun Frigate (elite crew) (17 pts)

The Battle of Lake Erie:
Wind from the S, blowing a light breeze.

(a) Lawrence	20 gun Sloop (crack crew) (9 pts)
(a) Niagara	20 gun Sloop (elite crew) (12 pts)
(b) Lady Prevost	13 gun Brig (crack crew) (5 pts)
(b) Detroit	19 gun Sloop (crack crew) (7 pts)
(b) Q. Charlotte	17 gun Sloop (crack crew) (6 pts)

Wasp vs. Reindeer:
Wind from the S, blowing a light breeze.

(a) Wasp	20 gun Sloop (elite crew) (12 pts)
(b) Reindeer	18 gun Sloop (elite crew) (9 pts)

Constitution vs. Cyane and Levant:
Wind from the S, blowing a moderate breeze.

(a) Constitution	44 gun Corvette (elite crew) (24 pts)
(b) Cyane	24 gun Sloop (crack crew) (11 pts)
(b) Levant	20 gun Sloop (crack crew) (10 pts)

Pellew vs. Droits de L'Homme:
Wind from the N, blowing a gale.

(b) Indefatigable	44 gun Frigate (elite crew) (14 pts)
(b) Amazon	36 gun Frigate (crack crew) (14 pts)
(f) Droits L'Hom	74 gun Ship of the Line (average crew) (24 pts)

Algeciras:
Wind from the SW, blowing a moderate breeze.

(b) Caesar	80 gun Ship of the Line (crack crew) (31 pts)
(b) Pompee	74 gun Ship of the Line (crack crew) (27 pts)
(b) Spencer	74 gun Ship of the Line (crack crew) (26 pts)
(b) Hannibal	98 gun 3 Decker SOL (crack crew) (28 pts)
(s) Real-Carlos	112 gun 3 Decker SOL (green crew) (27 pts)
(s) San Fernando	96 gun 3 Decker SOL (green crew) (24 pts)
(s) Argonauta	80 gun Ship of the Line (green crew) (23 pts)
(s) San Augustine	74 gun Ship of the Line (green crew) (20 pts)
(f) Indomptable	80 gun Ship of the Line (average crew) (27 pts)
(f) Desaix	74 gun Ship of the Line (average crew) (24 pts)

Lake Champlain:
Wind from the N, blowing a fresh breeze.

(a) Saratoga	26 gun Sloop (crack crew) (12 pts)
(a) Eagle	20 gun Sloop (crack crew) (11 pts)
(a) Ticonderoga	17 gun Sloop (crack crew) (9 pts)
(a) Preble	7 gun Brig (crack crew) (4 pts)
(b) Confiance	37 gun Frigate (crack crew) (14 pts)
(b) Linnet	16 gun Sloop (elite crew) (10 pts)
(b) Chubb	11 gun Brig (crack crew) (5 pts)

Last Voyage of the USS President:
Wind from the N, blowing a fresh breeze.

(a) President	44 gun Frigate (elite crew) (24 pts)
(b) Endymion	40 gun Frigate (crack crew) (17 pts)
(b) Pomone	44 gun Frigate (crack crew) (20 pts)
(b) Tenedos	38 gun Frigate (crack crew) (15 pts)

Hornblower and the Natividad:

Wind from the E, blowing a gale.

A scenario for you Horny fans. Remember, he sank the Natividad against heavy odds and winds. Hint: don't try to board the Natividad, her crew is much bigger, albeit green.

(b) Lydia	36 gun Frigate (elite crew) (13 pts)
(s) Natividad	50 gun Ship of the Line (green crew) (14 pts)

Curse of the Flying Dutchman:

Wind from the S, blowing a fresh breeze.

Just for fun, take the Piece of cake.

(s) Piece of Cake	24 gun Corvette (average crew) (9 pts)
(f) Flying Dutchy	120 gun 3 Decker SOL (elite crew) (43 pts)

The South Pacific:

Wind from the S, blowing a strong breeze.

(a) USS Scurvy	136 gun 3 Decker SOL (mutinous crew) (27 pts)
(b) HMS Tahiti	120 gun 3 Decker SOL (elite crew) (43 pts)
(s) Australian	32 gun Frigate (average crew) (9 pts)
(f) Bikini Atoll	7 gun Brig (crack crew) (4 pts)

Hornblower and the battle of Rosas

Wind from the E, blowing a fresh breeze.

The only battle Hornblower ever lost. He was able to dismast one
ship and stern rake the others though. See if you can do as well.

(b) Sutherland	74 gun Ship of the Line (crack crew) (26 pts)
(f) Turenne	80 gun 3 Decker SOL (average crew) (27 pts)
(f) Nightmare	74 gun Ship of the Line (average crew) (24 pts)
(f) Paris	112 gun 3 Decker SOL (green crew) (27 pts)
(f) Napolean	74 gun Ship of the Line (green crew) (20 pts)

Cape Horn:

Wind from the NE, blowing a strong breeze.

(a) Concord	80 gun Ship of the Line (average crew) (27 pts)
(a) Berkeley	98 gun 3 Decker SOL (crack crew) (28 pts)
(b) Thames	120 gun 3 Decker SOL (elite crew) (43 pts)
(s) Madrid	112 gun 3 Decker SOL (green crew) (27 pts)
(f) Musket	80 gun 3 Decker SOL (average crew) (27 pts)

New Orleans:

Wind from the SE, blowing a fresh breeze.

Watch that little Cypress go!

(a) Alligator	120 gun 3 Decker SOL (elite crew) (43 pts)
(b) Firefly	74 gun Ship of the Line (crack crew) (27 pts)
(b) Cypress	44 gun Frigate (elite crew) (14 pts)

Botany Bay:

Wind from the N, blowing a fresh breeze.

(b) Shark	64 gun Ship of the Line (average crew) (18 pts)
(f) Coral Snake	44 gun Corvette (elite crew) (24 pts)
(f) Sea Lion	44 gun Frigate (elite crew) (24 pts)

Voyage to the Bottom of the

Wind from the NW, blowing a fresh breeze.

This one is dedicated to Richard Basehart and David Hedison.

(a) Seaview	120 gun 3 Decker SOL (elite crew) (43 pts)
(a) Flying Sub	40 gun Frigate (crack crew) (17 pts)
(b) Mermaid	136 gun 3 Decker SOL (mutinous crew) (27 pts)
(s) Giant Squid	112 gun 3 Decker SOL (green crew) (27 pts)

Frigate Action:

Wind from the E, blowing a fresh breeze.

(a) Killdeer	40 gun Frigate (average crew) (15 pts)
(b) Sandpiper	40 gun Frigate (average crew) (15 pts)
(s) Curlew	38 gun Frigate (crack crew) (16 pts)

The Battle of Midway:

Wind from the E, blowing a moderate breeze.

(a) Enterprise	80 gun Ship of the Line (crack crew) (31 pts)
(a) Yorktown	80 gun Ship of the Line (average crew) (27 pts)
(a) Hornet	74 gun Ship of the Line (average crew) (24 pts)
(j) Akagi	112 gun 3 Decker SOL (green crew) (27 pts)
(j) Kaga	96 gun 3 Decker SOL (green crew) (24 pts)
(j) Soryu	80 gun Ship of the Line (green crew) (23 pts)

Star Trek:

Wind from the S, blowing a fresh breeze.

(a) Enterprise	450 gun Ship of the Line (elite crew) (75 pts)
(a) Yorktown	450 gun Ship of the Line (elite crew) (75 pts)
(a) Reliant	450 gun Ship of the Line (elite crew) (75 pts)
(a) Galileo	450 gun Ship of the Line (elite crew) (75 pts)
(k) Kobayashi Maru	450 gun Ship of the Line (elite crew) (75 pts)
(k) Klingon II	450 gun Ship of the Line (elite crew) (75 pts)
(o) Red Orion	450 gun Ship of the Line (elite crew) (75 pts)
(o) Blue Orion	450 gun Ship of the Line (elite crew) (75 pts)

IMPLEMENTATION

sail is really two programs in one. Each player starts up a process which runs his own ship. In addition, a driver process is forked (by the first player) to run the computer ships and take care of global bookkeeping.

Because the driver must calculate moves for each ship it controls, the more ships the computer is playing, the slower the game will appear.

If a player joins a game in progress, he will synchronize with the other players (a rather slow process for everyone), and then he may play along with the rest.

To implement a multi-user game, the communicating processes must use a common temporary file as a place to read and write messages. In addition, a locking mechanism must be provided to ensure exclusive access to the shared file. For example, **sail** uses a temporary file named **/tmp/#sailsink.21** for scenario 21, and corresponding file names for the other scenarios. To provide exclusive access to the temporary file, **sail** uses a technique stolen from an old game called "pubcaves" by Jeff Cohen. Processes do a busy wait in the loop

```
for (n = 0; link(sync_file, sync_lock) < 0 && n < 30; n++)
        sleep(2);
```

until they are able to create a link to a file named **/tmp/#saillock.??**. The "??" correspond to the scenario number of the game. Since UNIX guarantees that a link will point to only one file, the process that succeeds in linking will have exclusive access to the temporary file.

CONSEQUENCES OF SEPARATE PLAYER AND DRIVER

When players do something of global interest, such as moving or firing, the driver must coordinate the action with the other ships in the game. For example, if a player wants to move in a certain direction, he writes a message into the temporary file requesting the driver to move his ship. Each "turn," the driver reads all the messages sent from the players and decides what happened. It then writes back into the temporary file new values of variables, etc.

The most noticeable effect this communication has on the game is the delay in moving. Suppose a player types a move for his ship and hits return. What happens then? The player process saves up messages to be written to the temporary file in a buffer. Every 7 seconds or so, the player process gets exclusive access to the temporary file and writes out its buffer to the file. The driver, running asynchronously, must read in the movement command, process it, and write out the results. This takes two exclusive accesses to the temporary file. Finally, when the player process gets around to doing another 7 second update, the results of the move are displayed on the screen. Hence, every movement requires four exclusive accesses to the temporary file (anywhere from 7 to 21 seconds depending upon asynchrony) before the player sees the results of his moves.

In practice, the delays are not as annoying as they would appear. There is room for "pipelining" in the movement. After the player writes out a first movement message, a second movement command can then be issued. The first message will be in the temporary file waiting for the driver, and the second will be in the file buffer waiting to be written to the file. Thus, by always typing moves a turn ahead of the time, the player can sail around quite quickly.

If the player types several movement commands between two 7 second updates, only the last movement command typed will be seen by the driver. Movement commands within the same update "overwrite" each other, in a sense.

NAME

scramble – turn your screen into a scramble puzzle

SYNOPSIS

scramble [–den] [**–c** *num*] [**–i** *num*] [**–p** *sec*] [**–x** *size*] [**–y** *size*]
[*message-line* ...]

DESCRIPTION

scramble turns your bitmap screen into a scrambling puzzle. If you solve the puzzle, you reclaim control of the screen. You can also quit by typing <EXIT> or CTRL/Q.

If you specify any *message-line* arguments, **scramble** displays the text in the empty square, one line per *message-line*. To include spaces in a single *message-line*, use backslash (\) or double-quote (") in the conventional manner.

Note: It is not nice to **crp**(1) or **rlogin**(1) onto other people's nodes and run this, since it scrambles their screens while they're trying to do important work (wink, wink).

OPTIONS

–d	Run in the current DM window, not on the entire screen.
–e	If the scrambling is to stop, just exit when done; don't give the player a chance to solve it.
–n	Don't number the pieces.
–c *num*	Scramble *num* times, then stop.
–i *num*	Keep scrambling the puzzle until you hit a recognized key (see below), but scramble a minimum of *num* times (default 0).
–p *sec*	After you solve the puzzle, pause for *sec* seconds before exiting so you can admire your skill (default 0).
–x *size*	Make the puzzle *size* squares across (default 5).
–y *size*	Make the puzzle *size* squares high (default 4).

COMMANDS

<EXIT>	Quit **scramble**.
←, →, ↑, ↓	Move a square into the empty box from the right, the left, above, or below, respectively.
|←, →|	Move the entire row into the empty box from the right or left, respectively.
Boxed arrow keys	Move the empty box to the left, to the right, up, or down.
Mouse buttons	Clicking any mouse button with the cursor in a square shifts the entire row from that piece onward towards the empty square. If the cursor is in neither the row nor the column of the empty square, this has no effect.

NAME

snake, **snscore** – display chase game

SYNOPSIS

/usr/games/snake [**–w***n*] [**–l***n*]
/usr/games/snscore

DESCRIPTION

snake is a display-based game which must be played on a **termcap**(5) supported screen. The object of the game is to make as much money as possible without getting eaten by the snake. The **–l** and **–w** options allow you to specify the length and width of the field. By default the entire screen (except for the last column) is used.

You are represented on the screen by an I. The snake is 6 squares long and is represented by S's. The money is $, and an exit is #. Your score is posted in the upper left hand corner.

You can move around using the same conventions as **vi**(1), the **h**, **j**, **k**, and **l** keys work, as do the arrow keys. Other possibilities include:

sefc	These keys are like **hjkl** but form a directed pad around the **d** key.
HJKL	These keys move you all the way in the indicated direction to the same row or column as the money. This does *not* let you jump away from the snake, but rather saves you from having to type a key repeatedly. The snake still gets all his turns.
SEFC	Likewise for the upper case versions on the left.
ATPB	These keys move you to the four edges of the screen. Their position on the keyboard is the mnemonic, e.g. **P** is at the far right of the keyboard.
x	This lets you quit the game at any time.
p	Points in a direction you might want to go.
w	Space warp to get out of tight squeezes, at a price.
!	Shell escape
CTRL/Z	Suspend the snake game, on systems which support it. Otherwise an interactive shell is started up.

To earn money, move to the same square the money is on. A new $ will appear when you earn the current one. As you get richer, the snake gets hungrier. To leave the game, move to the exit (#).

A record is kept of the personal best score of each player. Scores are only counted if you leave at the exit, getting eaten by the snake is worth nothing.

As in pinball, matching the last digit of your score to the number which appears after the game is worth a bonus.

To see who wastes time playing snake, run /usr/games/snscore.

FILES

/usr/games/lib/snakerawscores	Database of personal bests
/usr/games/lib/snake.log	Log of games played
/usr/games/busy	Program to determine if system too busy

BUGS

When playing on a small screen, it's hard to tell when you hit the edge of the screen.

The scoring function takes into account the size of the screen. A perfect function to do this equitably has not been devised.

NAME

strfile, unstr – create a random access file for storing strings

SYNOPSIS

strfile [*options*] *sourcefile* [*datafile*]

unstr [**–o**] [**–c***x*] *datafile* [*outfile*]

DESCRIPTION

strfile converts a file containing a set of strings into a data file which contains those strings along with a seek pointer table to the beginning of each. This allows random access to the strings. **strfile**'s main use is to add entries to the **fortune**(6) database.

strfile creates *datafile* from a *sourcefile* that consists of strings separated by lines starting with "%%" or "%–". Anything following these characters on the line will be ignored, so comments can be placed on these lines. A "%%" simply separates strings; a "%–" separates not only strings but sections. A file can have up to four sections (in other words, up to three delimiters). This can be used in a program-defined way.

If you do not specify a *datafile* on the command line, **strfile** creates a file named *sourcefile*.**dat**. The *datafile* contains a header describing its contents, the seek pointers to the beginning of each string, and the strings themselves (terminated by null bytes).

The format of the header is

```
# define  MAXDELIMS 3

# define  STR_RANDOM                         0x1
# define  STR_ORDERED                        0x2

typedef struct {
          unsigned long  str_numstr;            /* # of strings in the file */
          unsigned long  str_longlen;           /* length of longest string */
          unsigned long  str_shortlen;          /* length of shortest string */
          long           str_delims[MAXDELIMS]; /* delimiter markings */
          off_t          str_dpos[MAXDELIMS];   /* delimiter positions */
          short          str_flags;             /* bit field for flags */
} STRFILE;
```

The values in **str_delims** are the indices of the first string which follows each "%–" in the file. The field **str_flags** has the bit str_random set if the **–r** flag was specified, or str_ordered if the **–o** flag was specified.

unstr undoes the work of **strfile**. It serves primarily as an emergency backup in case you accidentally delete your source file but still have your data file. **unstr** reads a data file and creates a corresponding output file of raw strings and delimiters.

You can invoke **unstr** with the name of the data file, the name of the output file, or both. If you specify both, **unstr** treats them literally as the input and output files. If you provide a single argument ending in ''.dat,'' **unstr** assumes this to be the data file, and writes its output to that filename stripped of the ''.dat'' suffix. If the single argument doesn't end in ''.dat,'' the program treats this as the name of the output file, and consequently reads its input from *outputfile*.dat.

If you want a character other than ''%'' as your delimiter, use the **–c** option to change it.

unstr normally prints out the strings in the order they occur in the data file. If you give it the **–o** option, it will write them out in the seek pointer order, which is different if the file was randomized or alphabetized when created. Using this option, you can created sorted versions of your input file by using **strfile –o**, and then using **unstr –o** to dump them out in the table order.

OPTIONS

Following are the options for **strfile**. Only the **–o** and **–c** options may be used with **unstr**.

–	Print a usage summary.
–c*x*	Use character *x* as the delimiter instead of %.
–s	Run silently; do not summarize processing at the end of the run.
–v	Use verbose mode; summarize processing at the end of the run (default).
–o	Order the strings alphabetically. **strfile** stores the strings in the same order in the data file as in the source, but the seek pointer table will be sorted in alphabetical order of the strings pointed to. Any initial non-alphanumeric characters are ignored. This sets the str_ordered bit in the str_flags field of the header.
–i	Ignore case when ordering.
–r	Randomize the order of the seek pointers in the table. The strings will be stored in the same order in *datafile* as in *sourcefile*, but the seek pointer table will be randomized. This sets the str_random bit in the str_flags field of the header.

EXAMPLES

To convert a file called scene which consists of lines like

```
%%
Hofstadter's Law:
        It always takes longer than you expect, even when you take
        Hofstadter's Law into account.
%%
"It is bad luck to be superstitious."
                    -- Andrew W. Mathis
%%
If A = B and B = C, then A = C, except where void or prohibited by law.
                    -- Roy Santoro
```

use the following command:

```
$ strfile scene
"scene" converted to "scene.dat"
There were 1168 strings
Longest string: 1156 bytes
Shortest string: 0 bytes
```

FILES

strfile.h	Header file

SEE ALSO

fortune(6)

NAME

teachgammon – teach the game of backgammon

SYNOPSIS

/usr/games/teachgammon

DESCRIPTION

This program teaches you the rules for backgammon, supplies hints on strategy, and provides a tutorial consisting of a practice game against the computer.

FILES

/etc/**termcap**	Terminal capability database

SEE ALSO

backgammon(6)

NAME

trek – trekkie game

SYNOPSIS

/usr/games/**trek** [[–a] *file*]

DESCRIPTION

trek is a game of space glory and war.

If a filename is given, a log of the game is written into that file. If the –a flag is given before the filename, that file is appended to, not truncated.

trek asks what length game you would like. Valid responses are ''short'', ''medium'', and ''long''. You may also type ''restart'', which restarts a previously saved game. The program then prompts for the skill level, to which you must respond ''novice'', ''fair'', ''good'', ''expert'', ''commodore'', or ''impossible''. You should normally start out with ''novice'' and work up.

In general throughout the game, if you forget what is appropriate the game will tell you what it expects if you type a question mark.

COMMAND SUMMARY

abandon
capture
cloak up/down
computer request; ...
damages
destruct
dock
help
impulse course distance
lrscan
move course distance
phasers automatic amount
phasers manual amt1 course1 spread1 ...
torpedo course [yes] angle/no
ram course distance
rest time
shell
shields up/down
srscan [yes/no]
status
terminate yes/no
undock
visual course
warp warp_factor

NAME

ttt – tic-tac-toe

SYNOPSIS

/usr/games/ttt

DESCRIPTION

ttt is the popular X and O game, but it is also a learning program that never makes the same mistake twice.

When you begin, the program prompts you with

Accumulated knowledge? (Yes or No)

You must respond with a yes or no. The program tells you how many "words" of knowledge it currently has in its learning file. Then **ttt** states whether or not this is a new game, prints a three-by-three grid of numbers 1-9, and prompts with "Your move?" as shown below:

```
new game

123
456
789
Your move?
```

As you specify numbers in the grid, the program places an "X" in the location where the number once was. After you move, the program takes a turn, placing an "O" in an available spot. This continues until either you or the program wins by creating a vertical, horizontal, or diagonal line of three of the same characters in a row on the grid. To exit the game, type an interrupt.

Although it learns, **ttt** learns slowly. It must lose nearly 80 games to completely know the game.

FILES

/usr/games/lib/ttt.a	Learning file

NAME

vine – grow vines

SYNOPSIS

/usr/games/vine [*options*]

DESCRIPTION

vine creates interesting growing things on your screen. To halt the kudzu-like onslaught, type an interrupt.

OPTIONS

–e	Grow vines along the edge of the window.
–c	Grow vines from the center of the window.
–h	Exit if the RETURN key is pressed.
–r	Display in reverse video.
–i[*num*]	Interleave black/white leaves, one black for every *num* white ones. If *num* is negative, perform white/black interleaving.
–s[*size*]	Set maximum size of leaves (default is 2).
–d *device*	Use alternate device. Possible values for *device* are **borrow** (use entire display), **borrow_nc** (same as **borrow** but doesn't clear screen first), **direct** (in current window only), and **bg** (use display background).
–v[*num*]	Grow *num* vines from the top of the window.
–b*num*	Specify *num* as probability of branching (default 5).
–l*num*	As *num* increases (to maximum 35), more leaves grow from each stem.
–R*num*	Specify *num* different leaf directions; rotate *num* degrees for each leaf.
–q	Display usage information.

EXAMPLE

An especially pretty liana can be generated with

```
vine –e –i2
```

CAUTION

The –c option creates immense numbers of forks, perhaps taking over your machine for lengthy periods.

NAME

worm – play the growing worm game

SYNOPSIS

/usr/games/worm [*size*]

DESCRIPTION

In **worm**, you are a little worm; your body is the string of ''o''s on the screen and your head is the ''@''. You move with the ''hjkl'' keys as in the game **snake**(6). If you don't press any keys, you continue in the direction you last moved. The uppercase ''HJKL'' keys move you as if you had pressed the corresponding lowercase key several times (9 for HL and 5 for JK) unless you run into a digit, in which case you stop.

On the screen you will see a digit. If your worm eats the digit, it will grow longer; the actual amount depends on the digit eaten. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into the sides of the screen or itself. The current score (how much the worm has grown) is shown in the upper left corner of the screen.

The optional argument, if present, sets the initial length of the worm.

BUGS

If the initial length of the worm is set to less than 1 or more than 75, various strange things happen.

NAME

worms – animate worms on a display terminal

SYNOPSIS

/usr/games/worms [*options*]

DESCRIPTION

worms generates squirming annelid-like creatures on your display. This is an adaptation of an older program called **WORM**.

OPTIONS

–field	Make a "field" for the worm(s) to eat.
–trail	Make each worm leave a trail behind it.
–length *l*	Make each worm *l* characters long.
–number *n*	Create *n* worms.

FILES

/etc/termcap

BUGS

The lower right-hand character position will not be updated properly on a terminal that wraps at the right margin.

Terminal initialization is not performed.

NAME

wump – the game of hunt-the-wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

wump plays the game of 'Hunt the Wumpus.' A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in *People's Computer Company, 2,* 2 (November 1973).

Reader's Response

Please take a few minutes to send us the information we need to revise and improve our manuals from your point of view.

Document Title: *SysV Command Reference*
Order No.: 005798–A00
Date of Publication: July, 1988

What type of user are you?

_____ System programmer; language ____________________	
_____ Applications programmer; language ________________	
_____ System maintenance person	_____ Manager/Professional
_____ System Administrator	_____ Technical Professional
_____ Student Programmer	_____ Novice
_____ Other	

How often do you use the Apollo system?____

What parts of the manual are especially useful for the job you are doing?____________________

What additional information would you like the manual to include?_________________________

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible. Specify additional index entries.)____________________

Your Name ______________________________ Date

Organization

Street Address

City ____________ State ____________ Zip

No postage necessary if mailed in the U.S.

cut or fold along dotted line

fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 78 CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER INC.
Technical Publications
P.O. Box 451
Chelmsford, MA 01824

fold

Reader's Response

Please take a few minutes to send us the information we need to revise and improve our manuals from your point of view.

Document Title: *SysV Command Reference*
Order No.: 005798-A00
Date of Publication: July, 1988

What type of user are you?

_____ System programmer; language ________________________
_____ Applications programmer; language ____________________
_____ System maintenance person _____ Manager/Professional
_____ System Administrator _____ Technical Professional
_____ Student Programmer _____ Novice
_____ Other

How often do you use the Apollo system?________________________________

What parts of the manual are especially useful for the job you are doing?____________________

What additional information would you like the manual to include?____________________

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible. Specify additional index entries.)____________________

Your Name Date

Organization

Street Address

City State Zip

No postage necessary if mailed in the U.S.

cut or fold along dotted line

fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS   PERMIT NO. 78   CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER INC.
Technical Publications
P.O. Box 451
Chelmsford, MA 01824

fold