NAME

intro — introduction to commands

DESCRIPTION

This section describes publicly accessible commands in alphabetic order. Certain distinctions of purpose are made in the headings:

(1) Commands of general utility.

(1C) Commands for communication with other systems.

(1G) Commands used primarily for graphics and computer-aided design.

N.B.: Commands related to system maintenance used to appear in section 1 manual pages and were distinguished by (1M) at the top of the page. These manual pages now appear in section 8.

SEE ALSO

Section (6) for computer games.

How to get started, in the Introduction.

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* and *exit*(2). The former byte is 0 for normal termination, the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously 'exit code', 'exit status' or 'return code', and is described only where special conventions are involved.

NAME

adb − debugger

SYNOPSIS

adb [−**w**] [−**k**] [-Idir] [objfil [corfil]]

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of *adb* cannot be used although the file can still be examined. The default for *objfil* is **a.out.** *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is **core.**

Requests to *adb* are read from the standard input and responses are to the standard output. If the −**w** flag is present then both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using *adb.*

The −**k** option makes *adb* do UNIX kernel memory mapping; it should be used when *core* is a UNIX crash dump or */dev/mem.*

The −**I** option specifies a directory where files to be read with $< or $<< (see below) will be sought; the default is */usr/lib/adb.*

Adb ignores QUIT; INTERRUPT causes return to the next *adb* command.

In general requests to *adb* are of the form

[*address*] [, *count*] [*command*] [;]

If *address* is present then *dot* is set to *address.* Initially *dot* is set to 0. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *Address* and *count* are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. If the operating system is being debugged either post-mortem or using the special file */dev/mem* to interactive examine and/or modify memory the maps are set to map the kernel virtual addresses which start at 0x80000000 (on the VAX). ADDRESSES.

EXPRESSIONS

. The value of *dot.*

\+ The value of *dot* incremented by the current increment.

^ The value of *dot* decremented by the current increment.

" The last *address* typed.

integer A number. The prefixes 0o and 0O ("zero oh") force interpretation in octal radix; the prefixes 0t and 0T force interpretation in decimal radix; the prefixes 0x and 0X force interpretation in hexadecimal radix. Thus 0o20 = 0t16 = 0x10 = sixteen. If no prefix appears, then the *default radix* is used; see the $d command. The default radix is initially hexadecimal. The hexadecimal digits are 0123456789abcdefABCDEF with the obvious values. Note that a hexadecimal number whose most significant digit would otherwise be an alphabetic character must have a 0x (or 0X) prefix (or a leading zero if the default radix is hexadecimal).

integer.fraction
A 32 bit floating point number.

'*cccc*' The ASCII value of up to 4 characters. \ may be used to escape a '.

< name

The value of *name*, which is either a variable name or a register name. *Adb* maintains a number of variables (see VARIABLES) named by single letters or digits. If *name* is a register name then the value of the register is obtained from the system header in *corfil*. The register names are those printed by the $r command.

symbol A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. The backslash character \ may be used to escape other characters. The value of the *symbol* is taken from the symbol table in *objfil*. An initial _ will be prepended to *symbol* if needed.

_ symbol

In C, the 'true name' of an external symbol begins with _. It may be necessary to utter this name to distinguish it from internal or hidden variables of a program.

routine.name

The address of the variable *name* in the specified C routine. Both *routine* and *name* are *symbols*. If *name* is omitted the value is the address of the most recently activated C stack frame corresponding to *routine*. (This form is currently broken on the VAX; local variables can be examined only with *dbx*(1).)

(*exp*) The value of the expression *exp*.

Monadic operators

**exp* The contents of the location addressed by *exp* in *corfil*.

@*exp* The contents of the location addressed by *exp* in *objfil*.

−*exp* Integer negation.

~*exp* Bitwise complement.

#*exp* Logical negation.

Dyadic operators are left associative and are less binding than monadic operators.

e1+*e2* Integer addition.

e1−*e2* Integer subtraction.

*e1***e2* Integer multiplication.

e1%*e2* Integer division.

e1&*e2* Bitwise conjunction.

e1|*e2* Bitwise disjunction.

e1#*e2* *E1* rounded up to the next multiple of *e2*.

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands '?' and '/' may be followed by '*'; see ADDRESSES for further details.)

?f Locations starting at *address* in *objfil* are printed according to the format *f*. *dot* is incremented by the sum of the increments for each format letter (q.v.).

/f Locations starting at *address* in *corfil* are printed according to the format *f* and *dot* is incremented as for '?'.

=f The value of *address* itself is printed in the styles indicated by the format *f*. (For **i** format '?' is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format *dot* is incremented by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows.

o	2	Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
O	4	Print 4 bytes in octal.
q	2	Print in signed octal.
Q	4	Print long signed octal.
d	2	Print in decimal.
D	4	Print long decimal.
x	2	Print 2 bytes in hexadecimal.
X	4	Print 4 bytes in hexadecimal.
u	2	Print as an unsigned decimal number.
U	4	Print long unsigned decimal.
f	4	Print the 32 bit value as a floating point number.
F	8	Print double floating point.
b	1	Print the addressed byte in octal.
c	1	Print the addressed character.
C	1	Print the addressed character using the standard escape convention where control characters are printed as ^X and the delete character is printed as ^?.
s	*n*	Print the addressed characters until a zero character is reached.
S	*n*	Print a string using the ^*X* escape convention (see **C** above). *n* is the length of the string including its zero terminator.
Y	4	Print 4 bytes in date format (see *ctime*(3)).
i	n	Print as machine instructions. *n* is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
a	0	Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.
	/	local or global data symbol
	?	local or global text symbol
	=	local or global absolute symbol
p	4	Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.
t	0	When preceded by an integer tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.
r	0	Print a space.
n	0	Print a newline.
"..."	0	Print the enclosed string.
^		*Dot* is decremented by the current increment. Nothing is printed.
+		*Dot* is incremented by 1. Nothing is printed.
−		*Dot* is decremented by 1. Nothing is printed.

newline

Repeat the previous command with a *count* of 1.

[?/]**l** *value mask*

Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If **L** is used then the match is for 4 bytes at a time instead of 2. If no match is found then *dot* is unchanged; otherwise *dot* is set to the matched location. If *mask* is omitted then −1 is used.

[?/]**w** *value* ...

Write the 2-byte *value* into the addressed location. If the command is **W**, write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m *b1 e1 f1*[?/]
New values for (*b1, e1, f1*) are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. If the '?' or '/' is followed by '*' then the second segment (*b2, e2, f2*) of the mapping is changed. If the list is terminated by '?' or '/' then the file (*objfil* or *corfil* respectively) is used for subsequent requests. (So that, for example, '/m?' will cause '/' to refer to *objfil.*)

> *name* *Dot* is assigned to the variable or register named.

! A shell (/bin/sh) is called to read the rest of the line following '!'.

$*modifier*
Miscellaneous commands. The available *modifiers* are:

- **<***f* Read commands from the file *f*. If this command is executed in a file, further commands in the file are not seen. If *f* is omitted, the current input stream is terminated. If a *count* is given, and is zero, the command will be ignored. The value of the count will be placed in variable *9* before the first command in *f* is executed.
- **<<***f* Similar to < except it can be used in a file of commands without causing the file to be closed. Variable *9* is saved during the execution of this command, and restored when it completes. There is a (small) finite limit to the number of << files that can be open at once.
- **>***f* Append output to the file *f*, which is created if it does not exist. If *f* is omitted, output is returned to the terminal.
- **?** Print process id, the signal which caused stoppage or termination, as well as the registers as **$r**. This is the default if *modifier* is omitted.
- **r** Print the general registers and the instruction addressed by **pc**. *Dot* is set to **pc**.
- **b** Print all breakpoints and their associated counts and commands.
- **c** C stack backtrace. If *address* is given then it is taken as the address of the current frame instead of the contents of the frame−pointer register. If **C** is used then the names and (32 bit) values of all automatic and static variables are printed for each active function. (broken on the VAX). If *count* is given then only the first *count* frames are printed.
- **d** Set the default radix to *address* and report the new value. Note that *address* is interpreted in the (old) current radix. Thus "10$d" never changes the default radix. To make decimal the default radix, use "0t10$d".
- **e** The names and values of external variables are printed.
- **w** Set the page width for output to *address* (default 80).
- **s** Set the limit for symbol matches to *address* (default 255).
- **o** All integers input are regarded as octal.
- **q** Exit from *adb.*
- **v** Print all non zero variables in octal.
- **m** Print the address map.
- **p** (*Kernel debugging*) Change the current kernel memory mapping to map the designated **user structure** to the address given by the symbol *_u*. The *address* argument is the address of the user's user page table entries (on the VAX).

:*modifier*
Manage a subprocess. Available modifiers are:

- **bc** Set breakpoint at *address*. The breakpoint is executed *count*−1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command is omitted or sets *dot* to zero then the breakpoint

causes a stop.

d Delete breakpoint at *address*.

r Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command.

c*s* The subprocess is continued with signal *s*, see *sigvec*(2). If *address* is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.

s*s* As for **c** except that the subprocess is single stepped *count* times. If there is no current subprocess then *objfil* is run as a subprocess as for **r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.

k The current subprocess, if any, is terminated.

VARIABLES

Adb provides a number of variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows.

0 The last value printed.
1 The last offset part of an instruction source.
2 The previous value of variable 1.
9 The count on the last $< or $<< command.

On entry the following are set from the system header in the *corfil*. If *corfil* does not appear to be a **core** file then these values are set from *objfil*.

b The base address of the data segment.
d The data segment size.
e The entry point.
m The 'magic' number (0407, 0410 or 0413).
s The stack segment size.
t The text segment size.

ADDRESSES

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1, e1, f1*) and (*b2, e2, f2*) and the *file address* corresponding to a written *address* is calculated as follows.

$$b1 \leqslant address < e1 \Rightarrow file\ address = address + f1 - b1, \text{ otherwise,}$$

$$b2 \leqslant address < e2 \Rightarrow file\ address = address + f2 - b2,$$

otherwise, the requested *address* is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. If a ? or / is followed by an * then only the second triple is used.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected then, for that file, *b1* is set to 0, *e1* is set to the maximum file size and *f1* is set to 0; in this way the whole file can be examined with no address translation.

FILES

a.out
core

SEE ALSO

cc(1), dbx(1), ptrace(2), a.out(5), core(5)

DIAGNOSTICS

'Adb' when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned nonzero status.

BUGS

Since no shell is invoked to interpret the arguments of the :r command, the customary wild-card and variable expansions cannot occur.

NAME

addbib — create or extend bibliographic database

SYNOPSIS

addbib [**−p** promptfile] [**−a**] database

DESCRIPTION

When this program starts up, answering "y" to the initial "Instructions?" prompt yields directions; typing "n" or RETURN skips them. *Addbib* then prompts for various bibliographic fields, reads responses from the terminal, and sends output records to a *database*. A null response (just RETURN) means to leave out that field. A minus sign (−) means to go back to the previous field. A trailing backslash allows a field to be continued on the next line. The repeating "Continue?" prompt allows the user either to resume by typing "y" or RETURN, to quit the current session by typing "n" or "q", or to edit the *database* with any system editor *(vi, ex, edit, ed)*.

The **−a** option suppresses prompting for an abstract; asking for an abstract is the default. Abstracts are ended with a CTRL-d. The **−p** option causes *addbib* to use a new prompting skeleton, defined in *promptfile*. This file should contain prompt strings, a tab, and the key-letters to be written to the *database*.

The most common key-letters and their meanings are given below. *Addbib* insulates you from these key-letters, since it gives you prompts in English, but if you edit the bibliography file later on, you will need to know this information.

%A	Author's name
%B	Book containing article referenced
%C	City (place of publication)
%D	Date of publication
%E	Editor of book containing article referenced
%F	Footnote number or label (supplied by *refer*)
%G	Government order number
%H	Header commentary, printed before reference
%I	Issuer (publisher)
%J	Journal containing article
%K	Keywords to use in locating reference
%L	Label field used by **−k** option of *refer*
%M	Bell Labs Memorandum (undefined)
%N	Number within volume
%O	Other commentary, printed at end of reference
%P	Page number(s)
%Q	Corporate or Foreign Author (unreversed)
%R	Report, paper, or thesis (unpublished)
%S	Series title
%T	Title of article or book
%V	Volume number
%X	Abstract — used by *roffbib*, not by *refer*
%Y,Z	ignored by *refer*

Except for 'A', each field should be given just once. Only relevant fields should be supplied. An example is:

%A	Bill Tuthill
%T	Refer — A Bibliography System
%I	Computing Services

%C Berkeley
%D 1982
%O UNX 4.3.5.

FILES

promptfile optional file to define prompting

SEE ALSO

refer(1), sortbib(1), roffbib(1), indxbib(1), lookbib(1)

AUTHORS

Al Stangenberger, Bill Tuthill

NAME

allusers – print list of all authorized users

SYNOPSIS

allusers

DESCRIPTION

Allusers prints a neatly formatted list of authorized users of this Unix system, giving login name and full name. Those accounts that the program can recognize as non-human are noted.

If you want the results sorted, you can type

```
allusers | sort
```

to get the list sorted alphabetically by user-name.

BUGS

Heuristic to recognize non-humans is simple-minded

Sorting by last name is hard to do, so this program can't.

NAME

altoload – load files from an Alto FTP "dump" format file

SYNOPSIS

altoload dumpfile

DESCRIPTION

The Alto FTP program can write collections of files into a single host file (called "dumping"), and can retrieve files from a dump file (called "loading".) The *altoload* program allows you to "load" *text files only* from a dump file.

The dump file must be on the Unix filesystem; this can be done either by using *ftp*(1) to retrieve a dump-format file from another host, or by using the Alto FTP program to write a dump file directly on the Unix filesystem.

When invoked with the name of the dump file as an argument, *altoload* will prompt you with the name of each component file, at which point you can do one of three things:

Type the name of a file to be written with the dump component file.

Type a newline; this will cause the a file to be written with the name of the component.

Type a '-'; this will cause the component file to be skipped.

All filenames are interpreted relative to the current working directory, and will be truncated to the usual maximum length.

SEE ALSO

ftp(1)

DIAGNOSTICS

Mostly self-explanatory (failure to open, read, or write a file). Input files in improper format will give error messages about "Unknown block types".

AUTHOR

Jeffrey Mogul

BUGS

Text files only.

Dump component files may have names much longer than 14 characters; rather than attempt to deal with this problem, the program simply askes the user to give every filename (if the default is used, the user is responsible for making sure that it is reasonable.)

NAME

ansi – read and write ANSI format magnetic tapes

SYNOPSIS

ansi [[command [arg ...]] ...]

DESCRIPTION

ansi reads, writes, and prints out the directories of ANSI standard labeled magnetic tapes. The program currently uses ASCII characters, and the "D" format, which allows tapes from most DEC computers to be read. Binary and/or EBCDIC modes would be easy to add.

The following commands are availale:

read The named file is read from the tape, and written onto a Unix file of the same name, but with uppercase converted to lower.

write The named files are written on the tape, with lower case in filenames converted to upper case. A tape must be initialized with a volume label before any files can be written on it.

init The tape is initialized, with the following argument giving the volume name. Please be careful with this option, since it effectively destroys any information that was previously on the tape.

load All of the files on the tape are read. This option would be unneccessry if wildcards could be passed unexpanded through the shell and handled by the "read" command.

pipe Sends output to standard output instead of the named file. Useful to rename a file on reading.

If no arguments are given, the files on the tape are listed.

AUTHOR

Bill Nowicki

SEE ALSO

ht (4) – the local magtape driver

DIAGNOSTICS

Most operations echo vital statistics like the number of files and blocks transfered. If a file name requested to be read is not found, that is so indicated. The program will not allow you to write files on a tape that has not been initialized, or write a file of the same name as one already on the tape.

BUGS

Currently handles only text files. Has not been tested with RT-11. The whole way that Berkeley did the tape handlers, encoding the rewindedness in the minor device number should be cleaned up. *Ioctl* calls should be used to do things like set density, rewind, and space forward and backward.

NAME

apl – apl interpreter

SYNPOSIS

apl [–m] [–e] [–q] [–r] [–t] [–c] [–C] [–d] [–D] [ws]
apl2 [–m] [–e] [–q] [–r] [–t] [–c] [–C] [–d] [–D] [ws]

DESCRIPTION

This is the Unix APL interpreter. It has lived through several different versions of Unix and grown steadily more complex. Currently, a version of APL for Unix on the PDP-11 and the VAX is supported. This version supports monadic and dyadic domino, a state indicator of sorts, and Unix I/O quad functions.

The best documentation concerning the use of APL once it has been started from the shell is the *Unix APL\11 User's Manual*. This manual includes a list of the APL character set, system commands, quad functions, and i-beam functions, as well as an overall description of the use of APL. The specifics are contained in the four appendices for easy reference by the more experienced user.

The command invoking APL may optionally contain the name of a workspace file to be loaded (default is "continue", or, if "continue" does not exist in the current directory, APL starts executing with a "clear ws").

There are all sorts of flags which may be specified when APL is invoked. Only a subset of these are of general usefulness; the remainder exist for convenience in debugging and software maintenance purposes. In the following description, the flags are presented from those which are of the most general interest to those which are of interest only to persons maintaining APL.

Normally, APL runs in "ASCII mode". (This is discussed more fully following the description of the various flags.) If "–m" is specified, APL "maps" the standard input and standard output as appropriate for use with an APL terminal.

By default, APL attempts to determine whether or not the standard input is a terminal. If not, all input will be echoed to the standard output. In this fashion, when APL is run with a pipe or disc file as input, the output clearly shows the commands issued along with their results. The "–e" flag forces APL to echo its input to its output regardless of the input device. Similarly, "–q" ("quiet") forces APL not to echo its input to the standard output.

The flag "–r" has meaning only when the Purdue EE editor XED is used. This flag is passed by APL to XED to invoke funny XED stuff. This is generally a non-portable feature.

By default, APL places its scratch files into /tmp. If the "–t" flag is specified, temporary files will be placed into the current directory.

By default, APL catches fatal signals (e.g. memory fault, floating-point exception, etc.) and prints a termination message of the form:

fatal signal: message

It then exits normally. If the flag "–c" or "–C" is specified, it will print this error message and then exit via an "abort", producing a core dump. If the flag "–d" or "–D" is specified, it will not catch fatal errors, and thus will be automatically terminated by the Unix kernel if a fatal signal is received. (This will also invoke a core dump.) These flags are useful for debugging APL, but aren't of much use to the ordinary user.

The program "apl2" is identical to "apl" except that "apl" is double-precision and "apl2" is single-precision. Workspaces are stored in whatever precision is in use, and are converted if necessary automatically when they are ")load"ed. Effectively, "apl2" has twice as much space in its internal workspace.

APL is designed to operate principally from ASCII terminals. Upper-case letters are used for the various APL symbols, as described in a separate document. Overstrike characters, which generally will not appear as overstruck characters on a CRT screen, are generated by typing the first character, a control-H, and the second character. The order of the two characters is not significant. The workspace used by APL is stored in this special ASCII format.

APL does support APL terminals. To use APL from an APL terminal, it is necessary to specify the "-m" flag when calling APL from the shell; this causes the APL character set to be mapped to/from ASCII for input/output. The workspace file is still stored in ASCII format; thus work may be done interchangeably on both types of terminals.

HISTORY

APL was originally written at Bell Labs by Ken Thompson, sometime before version six Unix. It was modified for a while at Yale University, and then came to Purdue University, where it has undergone extensive modification. It is currently being supported by the Electrical Engineering Unix network. Complaints, suggestions, or whatever should be forwarded to user "bruner" on the EE Network system, or sent to either John Bruner or Dr. Anthony P. Reeves in the school of Electrical Engineering at Purdue University.

FILES

/tmp/apled.###### - editor temporary file
/tmp/aplws.###### - workspace temporary file
continue - default workspace file

SEE ALSO

aplcvt(1) – convert between PDP-11 and VAX workspace formats
aplopr(1) – output APL files to the Printronix printer
cata(1) – display functions with APL line numbers
prws(1) – print workspace

BUGS

Character comparisons do not work.
Only a restricted form of dyadic format is available. Laminate is not supported.
The workspace size on the PDP-11 is limited to about 5000 items in APL and 10000 in APL2.
The workspace size on the VAX is limited only by the virtual memory system.

NAME

apply − apply a command to a set of arguments

SYNOPSIS

apply [−**a***c*] [−*n*] command args ...

DESCRIPTION

Apply runs the named *command* on each argument *arg* in turn. Normally arguments are chosen singly; the optional number *n* specifies the number of arguments to be passed to *command*. If *n* is zero, *command* is run without arguments once for each *arg*. Character sequences of the form %*d* in *command*, where *d* is a digit from 1 to 9, are replaced by the *d*'th following unused *arg*. If any such sequences occur, *n* is ignored, and the number of arguments passed to *command* is the maximum value of *d* in *command*. The character '%' may be changed by the −**a** option.

Examples:

```
apply echo *
```

is similar to ls(1);

```
apply −2 cmp a1 b1 a2 b2 ...
```

compares the 'a' files to the 'b' files;

```
apply −0 who 1 2 3 4 5
```

runs who(1) 5 times; and

```
apply 'ln %1 /usr/joe' *
```

links all files in the current directory to the directory /usr/joe.

SEE ALSO

sh(1)

AUTHOR

Rob Pike

BUGS

Shell metacharacters in *command* may have bizarre effects; it is best to enclose complicated commands in single quotes ' '.

There is no way to pass a literal '%2' if '%' is the argument expansion character.

NAME

apropos — locate commands by keyword lookup

SYNOPSIS

apropos keyword ...

DESCRIPTION

Apropos shows which manual sections contain instances of any of the given keywords in their title. Each word is considered separately and case of letters is ignored. Words which are part of other words are considered thus looking for compile will hit all instances of 'compiler' also. Try

apropos password

and

apropos editor

If the line starts 'name(section) ...' you can do 'man section name' to get the documentation for it. Try 'apropos format' and then 'man 3s printf' to get the manual on the subroutine *printf.*

Apropos is actually just the **−k** option to the *man*(1) command.

FILES

/usr/lib/whatis data base

SEE ALSO

man(1), whatis(1), catman(8)

AUTHOR

William Joy

NAME

ar — archive and library maintainer

SYNOPSIS

ar key [posname] afile name ...

DESCRIPTION

Ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the loader. It can be used, though, for any similar purpose. **N.B:** This version of *ar* uses a ASCII-format archive which is portable among the various machines running UNIX. Programs for dealing with older formats are available: see *arcv*(8).

Key is one character from the set **drqtpmx,** optionally concatenated with one or more of **vuaibclo.** *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character **u** is used with **r,** then only those files with 'last-modified' dates 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 append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.

t Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p Print the named files in the archive.

m Move the 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 Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file. Normally the 'last-modified' date of each extracted file is the date when it is extracted. However, if **o** is used, the 'last-modified' date is reset to the date recorded in the archive.

v Verbose. Under the verbose option, *ar* 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,** it gives a long listing of all information about the files. When used with **p,** it precedes each file with a name.

c Create. Normally *ar* will create *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.

l Local. Normally *ar* places its temporary files in the directory /tmp. This option causes them to be placed in the local directory.

FILES

/tmp/v* temporaries

SEE ALSO

lorder(1), ld(1), ranlib(1), ar(5), arcv(8)

BUGS

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

The 'last-modified' date of a file will not be altered by the **o** option if the user is not the owner of the extracted file, or the super-user.

NAME

arptab – show contents of kernel ARP table

SYNOPSIS

arptab

DESCRIPTION

This program prints the entries in the kernel's ARP (Address Resolution Protocol) table. This is of interest only to network debuggers.

The format of the output is:
table-index: internet-address ethernet-address flags

AUTHOR

Greg Satz (satz@SRI-TSC)

NAME

as – VAX-11 assembler

SYNOPSIS

as [**–d124**] [**–L**] [**–W**] [**–V**] [**–J**] [**–R**] [**–t** directory] [**–o** objfile] [name ...]

DESCRIPTION

As assembles the named files, or the standard input if no file name is specified. The available flags are:

–d Specifies the number of bytes to be assembled for offsets which involve forward or external references, and which have sizes unspecified in the assembly language. The default is **–d4.**

–L Save defined labels beginning with a 'L', which are normally discarded to save space in the resultant symbol table. The compilers generate such temporary labels.

–V Use virtual memory for some intermediate storage, rather than a temporary file.

–W Do not complain about errors.

–J Use long branches to resolve jumps when byte-displacement branches are insufficient. This must be used when a compiler-generated assembly contains branches of more than 32k bytes.

–R Make initialized data segments read-only, by concatenating them to the text segments. This obviates the need to run editor scripts on assembly code to make initialized data read-only and shared.

–t Specifies a directory to receive the temporary file, other than the default /tmp.

All undefined symbols in the assembly are treated as global.

The output of the assembly is left on the file *objfile;* if that is omitted, *a.out* is used.

FILES

/tmp/as*	default temporary files
a.out	default resultant object file

SEE ALSO

ld(1), nm(1), adb(1), dbx(1), a.out(5)
Auxiliary documentation *Assembler Reference Manual.*

AUTHORS

John F. Reiser
Robert R. Henry

BUGS

–J should be eliminated; the assembler should automatically choose among byte, word and long branches.

NAME

as68 — .a68 -> .b assembler component of cc68

SYNOPSIS

as68 [-godspel] filename

DESCRIPTION

As68 is the 68000 assembler. The input is taken from filename.a68, if present, otherwise from filename. The output is sent to filename.b. More than one input file can be specified, but only a single output is generated. The available flags are

-g Undefined symbols are automatically declared global for later resolution by the loader.

-o filename
Direct output to filename.

-d Print info helpful for debugging the assembler

-s Put symbol table in list.out (relocatable values only)

-p Print listing on stdout

-e External symbols only in output

-l produces a listing, filename.list

FILES

/usr/sun/a68 /usr/bin/as68 /usr/sun/doc/a68opcodes

SEE ALSO

cc68 (1), pc68(1), ld68 (1).

NAME

at − execute commands at a later time

SYNOPSIS

at time [day] [file]

DESCRIPTION

At squirrels away a copy of the named *file* (standard input default) to be used as input to *sh*(1) (or *csh*(1) if you normally use it) at a specified later time. A *cd* command to the current directory is inserted at the beginning, followed by assignments to all environment variables (excepting the variable TERM, which is useless in this context.) When the script is run, it uses the user and group ID of the creator of the copy file.

The *time* is 1 to 4 digits, with an optional following 'A', 'P', 'N' or 'M' for AM, PM, noon or midnight. One and two digit numbers are taken to be hours, three and four digits to be hours and minutes. If no letters follow the digits, a 24 hour clock time is understood.

The optional *day* is either (1) a month name followed by a day number, or (2) a day of the week; if the word 'week' follows invocation is moved seven days further off. Names of months and days may be recognizably truncated. Examples of legitimate commands are

```
at 8am jan 24
at 1530 fr week
```

At programs are executed by periodic execution of the command */usr/lib/atrun* from *cron*(8). The granularity of *at* depends upon how often *atrun* is executed.

Standard output or error output is lost unless redirected.

FILES

/usr/lib/atrun	executor (run by *cron*(8)).
in /usr/spool/at:	
yy.ddd.hhhh.*	activity for year yy, day dd, hour hhhh.
lasttimedone	last *hhhh*
past	activities in progress

SEE ALSO

calendar(1), pwd(1), sleep(1), cron(8)

DIAGNOSTICS

Complains about various syntax errors and times out of range.

BUGS

Due to the granularity of the execution of */usr/lib/atrun,* there may be bugs in scheduling things almost exactly 24 hours into the future.

NAME

awk − pattern scanning and processing language

SYNOPSIS

awk [**−F***c*] [prog] [file] ...

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*.

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, *vide infra*.) 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, newlines 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 *>file* 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)).

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*. 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 a 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 newline); 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 }
```

SEE ALSO

lex(1), sed(1)

A. V. Aho, B. W. Kernighan, P. J. Weinberger, *Awk — a pattern scanning and processing language*

BUGS

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 "" to it.

NAME

backup — make a backup version copy of a file

SYNOPSIS

backup [−q] [−w] [−u] files ...

DESCRIPTION

This command is used to generate a backup version of a file, according to the convention we have adopted: backup versions have the original filename, truncated if necessary, with the characters ".v" and a digit string appended. The version numbers run in increasing order, starting at one.

For example, given the input filename "program.c", *backup* will copy it to "program.c.v1" if no file with that name exists already. Otherwise, it will attempt successively higher version numbers until it finds an unused one.

The resultant copy will normally have the same access mode as the original, except that *no* users will have write access. If the access mode of the original file cannot be determined, the resultant file will have a mode of 0444 (r−−r−−r−−); that is, read access to all, only.

If all goes well, the name and access mode of the new file will be reported on the standard output, unless the **−q** (quiet) option is given.

The following options are recognized:

-q	(Quiet) -- do not report anything on the standard output (errors still go to the standard error stream.
-w	(allow Write) -- do not remove write access from backup file; that is, the backup file gets the same access that the original had.
-u	(Unsafe) -- instead of using the ".vnn" convention, append a ".bak" to the filename to get the backup filename. Moreover, allow the new backup to overwrite an existing file of the same name. This is useful to prevent a proliferation of useless backups. You are advised if an old file was overwritten.

SEE ALSO

backupname (1), cp (1)

AUTHOR

Jeffrey Mogul

DIAGNOSTICS

Most are self-explanatory; also, this program inherits most of the diagnostics generated by *cp*(1) , if there are any.

The program will complain when asked to backup a file which exists and is a directory (e.g., either of "/usr/bin" or "/usr/bin/"). The program will likewise complain if it asked to make a backup of a special file.

The number of possible versions is limited to the highest positive signed integer; the program will complain if it cannot create a unique name within this range. Don't expect this to happen too soon.

BUGS

A design decision was made not to strip off existing version suffixes before tacking on a new one; thus, "name.c.v3" becomes "name.c.v3.v1". Intelligent use would be to only apply the program to the original filename; this is not a completely useful version numbering system as in TOPS-20 or VMS.

NAME

basename — strip filename affixes

SYNOPSIS

basename string [suffix]

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 ` ` in shell procedures.

This shell procedure invoked with the argument */usr/src/bin/cat.c* compiles the named file and moves the output to *cat* in the current directory:

```
cc $1
mv a.out `basename $1 .c`
```

SEE ALSO

sh(1)

NAME

bboard – bulletin board reading program

SYNOPSIS

bboard [–fhlpq] [number **] [** –number **]**

DESCRIPTION

Bboard is used to read bulletin board messages. When run, the program engages in an interactive dialog in which it presents you with a summary of each bboard message and asks you if you want to see it. These messages are put on the bboard by means of mailing to the user 'bboard'.

Bboard is normally used automatically at login by placing a call to it in your *.login* (*.profile* if you use */bin/sh*). However, you can run it whenever you like, to see messages that have been posted since you last logged in. When run, it will prompt you with the source and subject of each new message and ask you if you would like to see the rest. (If there is no subject line, the first few non-blank lines of the message will be displayed instead.) The possible responses are:

y or just **RETURN** type the rest of the message

n or + skip this message and go on to the next message.

– back up to the previous message and ask again.

q Answers "no" for this message, then drops you out of *bboard;* the next time you run *bboard* it will pick up where you left off.

s append the current message to the file "Messages" in the current directory; An 's' can be followed by a space and a filename to receive the message instead of the default "Messages".

m causes a copy of the specified message to be placed in a scratch mailbox and then *mail*(1) to be invoked on that mailbox.

Bboard keeps track of the next message you will see by a number in the file *.bbrc* in your home directory. In the directory */usr/spool/bboard* it keeps a set of files whose names are the (sequential) numbers of the messages they represent. The file */usr/spool/bboard/bounds* shows the low and high number of the messages in the directory so that *bboard* can quickly determine if there are no messages for you.

Command line options to **Bboard** include

–f which causes it to be silent instead of saying "No new messages" if there aren't any.

–q Queries whether there are messages, printing "There are new messages." if there are. The command "bboard –q" is often used in login scripts.

–h causes *bboard* to print the header (first part) of messages only.

–l option causes only locally originated messages to be reported.

num A message number can be given on the command line, causing *bboard* to start at the specified message rather than at the next message indicated by your *.bbrc* file. Thus

bboard –h 1

prints the first part of all messages.

–number

will cause *bboard* to start *number* messages back from the one indicated by your *.bbrc* file, useful for reviews of recent messages.

–m prevents the usual behavior of piping long messages through *more*(1).

Within *bboard* you can also go to any specific message by typing its number when *bboard* requests input as to what to do.

FILES

/usr/spool/bboard/*	database
~/.bbrc	number of next message to be presented

AUTHORS

William Joy
David Wasley

SEE ALSO

mail(1), more(1), msgs(1)

BUGS

Does not properly interface to MH for the "m" command.

NAME

bc − arbitrary-precision arithmetic language

SYNOPSIS

bc [**−c**] [**−l**] [file ...]

DESCRIPTION

Bc is an interactive processor for a language which resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The **−l** argument stands for the name of an arbitrary precision math library. The syntax for *bc* programs is as follows; L means letter a-z, E means expression, S means statement.

Comments
are 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.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines 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.

For 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.

Bc is actually a preprocessor for *dc*(1), which it invokes automatically, unless the **-c** (compile only) option is present. In this case the *dc* input is sent to the standard output instead.

FILES

/usr/lib/lib.b mathematical library
dc(1) desk calculator proper

SEE ALSO

dc(1)
L. L. Cherry and R. Morris, *BC - An arbitrary precision desk-calculator language*

BUGS

No &&, ||, or ! operators.
For statement must have all three E's.
Quit is interpreted when read, not when executed.

NAME

bibtex – make a LaTeX bibliography

SYNOPSIS

bibtex auxname

DESCRIPTION

Bibtex reads the top-level auxilliary (.aux) file output during the running of *latex (1)* and creates a bibliography (.bbl) file that can be included in the LaTeX source file. The auxname on the command line should be given without an extension. Each \cite in the source file is looked up in bibliography files to gather together those used in the document. Then a bibliography style file is executed to write a \thebibliography environment.

The source file should have defined the bibliography (.bib) files to search with the \bibliography command, and the bibliography style (.bst) file to execute with the \bibstyle command. *Bibtex* incorporates the path searching mechanism described in the manual page for *tex (1)*. It searches the TEXINPUTS path for .aux, .bbl, and .bst files.

The bibliography file format is a subset of that allowed in Scribe bibliographies. Only the delimiter pairs {...} and "..." are allowed inside entries. Entries themselves can be delimited by (...) also. The = sign between field names and field values is not optional.

It is intended that there will eventually be bibliography style files capables of handling all the entry types and field names allowed in Scribe bibliographies. At the moment, there is only a very preliminary style file called cacm.bst, available in the standard TeX directory.

SEE ALSO

latex(1), tex(1).

BUGS

The cacm style file is really just a skeleton so far.

HISTORY

Written by Oren Patashnik under the direction of Leslie Lamport. Ported to UNIX by Howard Trickey, June 1984.

NAME

biff − be notified if mail arrives and who it is from

SYNOPSIS

biff [yn]

DESCRIPTION

Biff informs the system whether you want to be notified when mail arrives during the current terminal session. The command

biff y

enables notification; the command

biff n

disables it. When mail notification is enabled, the header and first few lines of the message will be printed on your screen whenever mail arrives. A "biff y" command is often included in the file *.login* or *.profile* to be executed at each login.

Biff operates asynchronously. For synchronous notification use the MAIL variable of *sh*(1) or the *mail* variable of *csh*(1).

SEE ALSO

csh(1), sh(1), mail(1), comsat(8C)

NAME

binmail − send or receive mail among users

SYNOPSIS

/bin/mail [**+**] [**−i**] [person] ...
/bin/mail [**+**] [**−i**] **−f** file

DESCRIPTION

Note: This is the old version 7 UNIX system mail program. The default *mail* command is described in *Mail*(1), and its binary is in the directory */usr/ucb*.

mail with no argument prints a user's mail, message-by-message, in last-in, first-out order; the optional argument **+** displays the mail messages in first-in, first-out order. For each message, it reads a line from the standard input to direct disposition of the message.

newline
: Go on to next message.

d
: Delete message and go on to the next.

p
: Print message again.

−
: Go back to previous message.

s [*file*] ...
: Save the message in the named *files* ('mbox' default).

w [*file*] ...
: Save the message, without a header, in the named *files* ('mbox' default).

m [*person*] ...
: Mail the message to the named *persons* (yourself is default).

EOT (control-D)
: Put unexamined mail back in the mailbox and stop.

q
: Same as EOT.

!*command*
: Escape to the Shell to do *command*.

*
: Print a command summary.

An interrupt normally terminates the *mail* command; the mail file is unchanged. The optional argument **−i** tells *mail* to continue after interrupts.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or a line with just '.') and adds it to each *person's* 'mail' file. The message is preceded by the sender's name and a postmark. Lines that look like postmarks are prepended with '>'. A *person* is usually a user name recognized by *login*(1). To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1C)).

The **−f** option causes the named file, for example, 'mbox', to be printed as if it were the mail file.

When a user logs in he is informed of the presence of mail.

FILES

/etc/passwd	to identify sender and locate persons
/usr/spool/mail/*	incoming mail for user *
mbox	saved mail
/tmp/ma*	temp file
/usr/spool/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

Mail(1), write(1), uucp(1C), uux(1C), xsend(1), sendmail(8)

BUGS

Race conditions sometimes result in a failure to remove a lock file.

Normally anybody can read your mail, unless it is sent by *xsend*(1). An installation can overcome this by making *mail* a set-user-id command that owns the mail directory.

NAME

boise -- send files to the HP2680a printer using TCP

SYNOPSIS

boise [options] [files]

DESCRIPTION

The *boise* program reads each file specified, prepends a header, and sends it off to the Boise printer. If no files are specified, *boise* sends its standard input to the printer.

Currently, the Boise printer can print text and DVI files. Please do not send it Press files. The program will recognize filename extensions ".dvi" and ".press" (currently, Press files are thrown away by the printer).

The available options include:

-b banner	Uses "banner" to label the output. It will appear on the cover page on the line labeled "File:".
-c n	Causes n copies of the output to be printed. The default is to print one copy.
-n name	Causes the given name to be used as the delivery address of your output (the "For user:" field on the cover sheet). This defaults to your full name.
-r	Causes the output to be rotated 90 degrees on the page (landscape mode). This is good for output that requires a wide page.
-v	Verbose mode. *Boise* will print a line on the standard error describing what was done with each file.
-w	Specifies that the files to be printed are in Waits (Sail) Ascii instead of standard Ascii.
-s server	Specifies what server the files are to be sent to. *Server* may be the name of any Internet host. The default is, of course, "boise".
-p port	Specifies which port on the server machine the files are to be sent to, in decimal. The default is 35. It is probably not useful to change this.
-m mode	Specifies a print mode other than default, where 1 is DVI, 2 is Press, and 3 is HP2680a.

SEE ALSO

pr(1), cat(1)

DIAGNOSTICS

Most are self-explanatory. The message "Connection timed out" means the printer server did not respond, probably because it is down.

NAME

btroff – troff to the ImPrint printer

SYNOPSIS

btroff [*troff* options] [flags] [*file* ...]

DESCRIPTION

Btroff runs *troff*(1) in an environment to produce typeset output on the Boise printer. It uses the *catboise* program to convert to DVI format.

Besides the *troff* flags, the following is recognized:

–c*n* Print *n* copies.

troff(1), boise(1), catboise(1).

BUGS

Not all characters are handled optimally. Italic spacing tends to be a little tight. Cut marks fall just outside the page boundaries, thus the first set of cut marks is lost and all subsequent ones fall at the bottom of the preceeding page. Backwards motion across page boundaries, as is sometimes done in *tbl*, will cause errors.

NAME

buildmake − preprocessor to provide extended syntax for makefiles

SYNOPSIS

buildmake [-f filename -Dx1=y1 -Dx2=y2 ... **]**

DESCRIPTION

Buildmake is a preprocessor that translates "buildfiles" into makefiles for use by the *make*(1) program. A buildfile has the same syntax as a makefile, with the addition of two features. A line of the form "#include *filename*" causes the named file to be inserted in the output, replacing the #include directive. An #ifdef/#else/#endif construct is also available, allowing sections of a buildfile to be conditionally included in the constructed makefile depending on whether it has defined a given symbol at the point the #ifdef is seen, as shown below.

```
NAME1=yes
#ifdef NAME1
This will be included in the output
#else NAME1
This will not be included
#endif NAME1
#ifdef NAME2
This will not be included unless NAME2 is defined elsewhere
#endif NAME2
```

The −D command line option is used to define symbols on the command line. The option -DNAME=value causes the line "NAME=value" to be inserted in the constructed makefile and causes NAME to be considered "defined" in subsequent #ifdef statements.

The −f option is used to specify the name of the input file, which defaults to "buildfile." The output file is always named "makefile."

SEE ALSO

make(1)

AUTHOR

Marvin Theimer, Stanford.

NAME

cal − print calendar

SYNOPSIS

cal [month] year

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. *Year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

BUGS

The year is always considered to start in January even though this is historically naive.
Beware that 'cal 78' refers to the early Christian era, not the 20th century.

NAME

calen – print large-format calendar

SYNOPSIS

calen [month] year [length] [–o]

DESCRIPTION

Calen prints a calendar, one page per month, on the standard output. You can pipe the output into a program such as *boise*(1) or *cz*(1) to get hardcopy, but since the output is 132 columns wide, you should use "landscape" mode (i.e., the –r flag) with either of these programs.

If only one numeric argument is specified, it is assumed to be the year; a calendar is generated for the entire year. If two numeric arguments are given, the first is taken to be a month, and the calendar is generated for that month only. If three numeric arguments are given, the last one specifies the number of months for which a calendar should be printed, starting with the month specified by the first two arguments.

A *month* argument should be between 1 (January) and 12 (December). A *year* argument should be between 1753 (the start of the Gregorian calendar) and 9999; if it is less than 100, then it is treated as part of the 20th century. For example, "84" corresponds to 1984.

If the –o flag is given, the output will contain backspaces and overstrikes to emphasize month and year. Note that neither Boise nor the Dover can handle this.

EXAMPLES

```
calen 84
```

prints a calendar for 1984

```
calen 6 84
```

prints a calendar for June, 1984

```
calen 7 1984 6
```

prints a calendar for July through December, 1984

AUTHOR

A. W. Rogers

NAME

calendar – reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

Calendar consults the file 'calendar' in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as 'Dec. 7,' 'december 7,' '12/7,' etc., are recognized, but not '7 December' or '7/12'. If you give the month as "*" with a date, i.e. "* 1", that day in any month will do. On weekends 'tomorrow' extends through Monday.

When an argument is present, *calendar* does its job for every user who has a file 'calendar' in his login directory and sends him any positive results by *mail*(1). Normally this is done daily in the wee hours under control of *cron*(8).

The file 'calendar' is first run through the "C" preprocessor, */lib/cpp*, to include any other calendar files specified with the usual "#include" syntax. Included calendars will usually be shared by all users, maintained and documented by the local administration.

FILES

calendar
/usr/lib/calendar to figure out today's and tomorrow's dates
/etc/passwd
/tmp/cal*
/lib/cpp, egrep, sed, mail as subprocesses

SEE ALSO

at(1), cron(8), mail(1)

BUGS

Calendar's extended idea of 'tomorrow' doesn't account for holidays.

NAME

cat − catenate and print

SYNOPSIS

cat [−u] [−n] [−s] [−v] file ...

DESCRIPTION

Cat reads each *file* in sequence and displays it on the standard output. Thus

cat file

displays the file on the standard output, and

cat file1 file2 >file3

concatenates the first two files and places the result on the third.

If no input file is given, or if the argument '−' is encountered, *cat* reads from the standard input file. Output is buffered in 1024-byte blocks unless the standard output is a terminal, in which case it is line buffered. The **−u** option makes the output completely unbuffered.

The **−n** option displays the output lines preceded by lines numbers, numbered sequentially from 1. Specifying the **−b** option with the **−n** option omits the line numbers from blank lines.

The **−s** option crushes out multiple adjacent empty lines so that the output is displayed single spaced.

The **−v** option displays non-printing characters so that they are visible. Control characters print like ^X for control-x; the delete character (octal 0177) prints as ^?. Non-ascii characters (with the high bit set) are printed as M- (for meta) followed by the character of the low 7 bits. A **−e** option may be given with the **−v** option, which displays a '$' character at the end of each line. Specifying the **−t** option with the **−v** option displays tab characters as ^I.

SEE ALSO

cp(1), ex(1), more(1), pr(1), tail(1)

BUGS

Beware of 'cat a b >a' and 'cat a b >b', which destroy the input files before reading them.

NAME

catboise – convert C/A/T files to DVI format and print on Boise

SYNOPSIS

catboise [options] [file]

DESCRIPTION

catboise converts *file*, which should be in C/A/T format (i.e. *troff* output), to DVI format for printing on the Boise printer. If no file name is given, the standard input is used. The options are:

- **−c***n* Prints *n* copies.
- **−b** The next argument is printed on the banner page to identify this listing.
- **−i** The next argument is the name of the output file.
- **−v** Produces more verbose output.
- **−d** Produces extensive output for debugging.
- **−C** The next argument is the name of a character mapping table. It will be assumed to be in a standard directory unless it begins with /. If not given, "catab" is used.

If the **−i** flag is not used, the output is written to a temporary file and, upon completion of processing, *boise*(1) is called to process the output of *catboise*. The **−v** and **−d** flags will be passed on to *boise* if specified.

Since the **−b** and **−c** flags are merely passed to *boise*, they have no effect if **−i** is used.

FILES

/tmp/dvi??????

SEE ALSO

boise(1), btroff(1)

BUGS

Not all symbols are handled properly.

Boxes generated by *tbl* sometimes are drawn over several pages, the text being filled in separately. This is impossible to do on a page printer without extensive sorting, which is not done.

Many minor problems occur because of the difference in resolution between the C/A/T phototypesetter and Boise.

AUTHORS

This program has been rewritten more times than you want to know.

NAME

catdvi – convert C/A/T files to DVI format

SYNOPSIS

catdvi [options] [file]

DESCRIPTION

catdvi converts *file*, which should be in C/A/T format (i.e. *troff* output), to DVI format for printing on an Imprint-10 printer. If no file name is given, the standard input is used. The options are:

- **−c***n* Prints *n* copies.
- **−b** The next argument is printed on the banner page to identify this listing.
- **−i** The next argument is the name of the output file.
- **−v** Produces more verbose output.
- **−d** Produces extensive output for debugging.
- **−C** The next argument is the name of a character mapping table. It will be assumed to be in a standard directory unless it begins with /. If not given, "catab" is used.

If the **−i** flag is not used, the output is written to a temporary file and, upon completion of processing, the input (C/A/T) file is unlinked and *dviimp* is called to process the output of *catdvi*. The **−v** and **−d** flags will be passed on to *dviimp* if specified.

Since the **−b** and **−c** flags are merely passed to *dviimp*, they have no effect if **−i** is used.

FILES

/tmp/dvi??????

SEE ALSO

dviimp(1), itroff(1)

BUGS

Not all symbols are handled properly.

Boxes generated by *tbl* sometimes are drawn over several pages, the text being filled in separately. This is impossible to do on a page printer without extensive sorting, which is not done.

Many minor problems occur because of the difference in resolution between the C/A/T phototypesetter and the IMPRINT-10.

AUTHOR

Imagen Corp.

NAME

cb − C program beautifier

SYNOPSIS

cb

DESCRIPTION

Cb places a copy of the C program from the standard input on the standard output with spacing and indentation that displays the structure of the program.

NAME

cc – C compiler

SYNOPSIS

cc [option] ... file ...

DESCRIPTION

Cc is the UNIX C compiler. *Cc* accepts several types of arguments:

Arguments whose names end with '.c' are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with '.o' substituted for '.c'. The '.o' file is normally deleted, however, if a single C program is compiled and loaded all at one go.

In the same way, arguments whose names end with '.s' are taken to be assembly source programs and are assembled, producing a '.o' file.

The following options are interpreted by *cc*. See *ld*(1) for load-time options.

−c Suppress the loading phase of the compilation, and force an object file to be produced even if only one program is compiled.

−g Have the compiler produce additional symbol table information for *dbx*(1). Also pass the **−lg** flag to *ld*(1).

−go Have the compiler produce additional symbol table information for the obsolete debugger *sdb*(1). Also pass the **−lg** flag to *ld*(1).

−w Suppress warning diagnostics.

−p Arrange for the compiler to produce code which counts the number of times each routine is called. If loading takes place, replace the standard startup routine by one which automatically calls *monitor*(3) at the start and arranges to write out a *mon.out* file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1).

−pg Causes the compiler to produce counting code in the manner of **−p**, but invokes a run-time recording mechanism that keeps more extensive statistics and produces a *gmon.out* file at normal termination. Also, a profiling library is searched, in lieu of the standard C library. An execution profile can then be generated by use of *gprof*(1).

−O Invoke an object-code improver.

−R Passed on to *as*, making initialized variables shared and read-only.

−S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed '.s'.

−E Run only the macro preprocessor on the named C programs, and send the result to the standard output.

−C prevent the macro preprocessor from eliding comments.

−o *output*
Name the final output file *output*. If this option is used the file 'a.out' will be left undisturbed.

−D*name=def*
−D*name*
Define the *name* to the preprocessor, as if by '#define'. If no definition is given, the name is defined as "1".

−U*name*
Remove any initial definition of *name*.

−I*dir* '*#*include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **−I** options, then in directories on a standard list.

−B*string*
Find substitute compiler passes in the files named *string* with the suffixes cpp, ccom and c2. If *string* is empty, use a standard backup version.

−t[p012]
Find only the designated compiler passes in the files whose names are constructed by a **−B** option. In the absence of a **−B** option, the *string* is taken to be '/usr/c/'.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier *cc* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **a.out.**

STANFORD MODIFICATIONS

The Stanford version of **cc** may add either or both of the following arguments to the default compilation sequence:

−I*/usr/stanford/include*
to the C preprocessor, which will include standard Stanford header files;

/usr/stanford/lib/libStanford.a
inserted before other libraries (including */lib/libc.a* and any debugging libraries) to the loader. The current implementation does not add *libStanford.a* to the default list of libraries, although this feature can become enabled. Two new options have been added to deal with these modifications:

−v This flag instructs **cc** to print, on the standard error, each pass of the compiler with its arguments. The user can use the **−v** flag to determine the default Stanford arguments.

−L This flag instructs **cc** not to use any default Stanford arguments; the compilation is done in a 'standard' environment.

FILES

file.c	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporary
/lib/cpp	preprocessor
/lib/ccom	compiler
/usr/c/occom	backup compiler
/usr/c/ocpp	backup preprocessor
/lib/c2	optional optimizer
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/usr/lib/gcrt0.o	startoff for gprof-profiling
/usr/stanford/lib/libStanford.a	standard Stanford library
/lib/libc.a	standard library, see *intro*(3)
/usr/lib/libc_p.a	profiling library, see *intro*(3)
/usr/include	standard directory for '*#*include' files
/usr/stanford/include	directory for Stanford '*#*include' files
mon.out	file produced for analysis by *prof*(1)
gmon.out	file produced for analysis by *gprof*(1)

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978
B. W. Kernighan, *Programming in C—a tutorial*
D. M. Ritchie, *C Reference Manual*
monitor(3), prof(1), gprof(1), adb(1), ld(1), dbx(1), as(1)

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader.

BUGS

The compiler currently ignores advice to put **char**, **unsigned char**, **short** or **unsigned short** variables in registers. It previously produced poor, and in some cases incorrect, code for such declarations.

NAME

cc68 – C compiler for the MC68000

SYNOPSIS

cc68 [option] ... file ...

DESCRIPTION

Cc68 is the UNIX C compiler modified for the MC68000. *Cc68* is a flexible program for translating between various types of files. The types catered for in order of appearance during translation are '.c' (C source files), '.s' (assembly language files), '.b' (relocatable binary files), 'b.out' (absolute binary files), '.r' (byte-reversed files, cf. *rev68(1)*). and '.dl' (Macsbug download format, cf. *dl68(1)*).

Arguments to cc68 are either flags or input files. The type of an input file is normally determined by its suffix. When an argument to cc68 is not a flag and has a suffix different from any of the above suffixes, it is assumed to be of one of the types '.c', '.b', or 'b.out', namely the latest of these three consistent with the type of the output (e.g. if the output type were '.s' or '.b' then the input would have to be '.c'). If it has no suffix it is assumed to be of type 'b.out'.

Translation proceeds as follows. Each '.c' and '.s' program is translated to a '.b' relocatable using cpp, ccom68, and as68 as necessary. Then all .b files including those produced by translation are link edited into the one file, called 'b.out'. If the only input file was a single '.c' program then the '.b' file is deleted, otherwise all '.b' files are preserved.

The amount of processing performed by cc68 may be decreased or increased with some of the options. The -S option takes translation no further than '.s' files, i.e. only cpp and ccom68 are applied. The -c option takes translation up to '.b' files, omitting the link editing and not deleting any '.b' files. The -d option goes beyond 'b.out' to produce a '.dl' file (using dl68) that may be downloaded by the Motorola MACSBUG monitor and the Sun1 monitor. The -r option similarly goes beyond 'b.out' to produce a '.r' file (using rev68) that may be loaded directly by 68000 code based on ld68. Both -d and -r may be used together.

The output may be named explicitly with the -o option; the output file's name should follow -o. Otherwise the name is 'b.out' in the normal case, or 'filename.dl' for the -d option, or 'filename.r' for the -r option, where 'filename' is the first '.c', '.s', or '.b' file named as an input. If the input is not in any of those three categories, the names 'd.out' and 'r.out' are used respectively for -d and -r.

The version of the target machine may be given as the flag **–v***n* where *n* is the version. The only recognized version at present is -vm, "Version Macsbug." The effect of giving the -vm flag is to add /usr/sun/dm/include to the include directories for cpp, to add /usr/sun/dm/lib as a library in which to look for -lx libraries, and to load the symbol table if any into the region starting at 0x6BA.

The file /usr/sun/lib/crt0.b is passed to ld68, ahead of all other .b files. This has the effect of defining the symbol _start to be at the text origin and having a routine that performs necessary initialization, enters main, and exits cleanly to the monitor.

The following options are interpreted by *cc*. See *ld68*(1) for load-time options.

–d Produce a .dl file suitable for downloading with the MACSBUG monitor of the Motorola Design Module, cf. *dl68*(1).

–r Produce a .r file suitable for direct loading by the 68000, cf. *rev68*(1).

–c Suppress the loading phase of the compilation, and force an object file to be produced even if only one program is compiled.

–w Suppress warning diagnostics. [Note: may not work.]

–O Invoke an object-code improver.

–S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed '.s'.

−E Run only the macro preprocessor on the named C programs, and send the result to the standard output.

−L Produce an assembly listing for each source file, with the suffixes changed to ".ls".

−R Preserve relocation commands in b.out.

−C prevent the macro preprocessor from eliding comments.

−V Link for a V kernel environment. This is equivalent to specifying **−i/usr/sun/lib/teamroot.b −T 10000** and **−lV** at the end.

−m Link for a Macintosh environment. This is equivalent to specifying **−i/usr/sun/lib/crtmac.b −T 0 −e _start −r −d** and **−lmac −lc** at the end.

−o *output*
Name the final output file *output*. If this option is used and the file 'b.out' already exists it will be left undisturbed.

−l*x* Include libx.a as a library ld68 should search in for undefined functions. x may be more than one letter, as in -lpup.

−T *org* Org specifies in hexadecimal where to begin loading the program.

−e *entrypoint*
Entrypoint specifies where to begin execution.

−D*name=def*
−D*name*
Define the *name* to the preprocessor, as if by '#define'. If no definition is given, the name is defined as "1".

−U*name*
Remove any initial definition of *name*.

−I*dir* '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **−I** options, then in directories on a standard list. The standard list is (in order of search) */usr/sun/include* and */usr/include*.

−B*string*
Find substitute compiler passes in the files named *string* with the suffixes cpp, ccom and c2. If *string* is empty, use a standard backup version. [Which doesn't work!]

−t[p012]
Find only the designated compiler passes in the files whose names are constructed by a **−B** option. In the absence of a **−B** option, the *string* is taken to be '/usr/c/'.

−−x By default, *cc68* passes a **−x** flag to *ld68*, in order to suppress local symbols from the final symbol table. The **−−x** flag inhibits this default.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier *cc68* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **b.out.**

FILES

file.c	input file
file.b	object file
b.out	loaded output
/tmp/ctm?	temporary
/lib/cpp	preprocessor
/usr/sun/c68/comp	compiler
/usr/sun/c68/o68	optional optimizer

/usr/sun/lib/crt0.b runtime startoff
/usr/sun/lib/libc.a standard library, see (3)
/usr/sun/include
/usr/include standard directories for '#include' files

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978
B. W. Kernighan, *Programming in C—a tutorial*
D. M. Ritchie, *C Reference Manual*
ld68(1)

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader.

BUGS

This is hacked up from *cc*(1), and probably could be improved.

NAME

ccom68 – .c -> .s translator component of cc68

SYNOPSIS

ccom68 [-lXp]

DESCRIPTION

Ccom68 is the UNIX C compiler modified for the MC68000. It takes its input from stdin and the resulting assembly code is printed on stdout.

The -l option generates line numbers in the output. The -p option causes the profile forming instruction "jbsr mcount" to be inserted at the entry to each function. The latter option must be preceded by X, which signals that it is a pass 2 option.

FILES

/usr/sun/c68 /usr/bin/ccom68

SEE ALSO

cc68 (1)

NAME

cd – change working directory

SYNOPSIS

cd directory

DESCRIPTION

Directory becomes the new working directory. The process 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. It is therefore recognized and executed by the shells. In *csh*(1) you may specify a list of directories in which *directory* is to be sought as a subdirectory if it is not a subdirectory of the current directory; see the description of the *cdpath* variable in *csh*(1).

SEE ALSO

csh(1), sh(1), pwd(1), chdir(2)

NAME

checknews – check to see if user has news

SYNOPSIS

checknews [**ynqevv**] [*readnews options*]

DESCRIPTION

checknews reports to the user whether or not he has news.

y Reports "There is news" if the user has news to read.

n Reports "No news" if there isn't any news to read.

q causes checknews to be quiet. Instead of printing a message, the exit status indicates news. A status of 0 means no news, 1 means there is news.

v alters the **y** message to show the name of the first newsgroup containing unread news. Doubling **v** (e.g. **vv**) will cause an explanation of *any* claim of new news, and is useful if **checknews** and **readnews** disagree on whether there is news.

e Executes *readnews*(1) if there is news.

If there are no options, **y** is the default.

FILES

~/.newsrc	Active newsgroups
/usr/lib/news/active	Options and list of previously read articles

SEE ALSO

inews(1), postnews(1), readnews(1)

NAME

checknr − check nroff/troff files

SYNOPSIS

checknr [**−s**] [**−f**] [**−a**.x1.y1.x2.y2. ... .xn.yn] [**−c**.x1.x2.x3 ... .xn] [*file* ...]

DESCRIPTION

Checknr checks a list of *nroff*(1) or *troff*(1) input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. If no files are specified, *checknr* checks the standard input. Delimeters checked are:

(1) Font changes using \fx ... \fP.

(2) Size changes using \sx ... \s0.

(3) Macros that come in open ... close forms, for example, the .TS and .TE macros which must always come in pairs.

Checknr knows about the *ms*(7) and *me*(7) macro packages.

Additional pairs of macros can be added to the list using the **−a** option. This must be followed by groups of six characters, each group defining a pair of macros. The six characters are a period, the first macro name, another period, and the second macro name. For example, to define a pair .BS and .ES, use **−a**.BS.ES

The **−c** option defines commands which would otherwise be complained about as undefined.

The **−f** option requests *checknr* to ignore \f font changes.

The **−s** option requests *checknr* to ignore \s size changes.

Checknr is intended to be used on documents that are prepared with *checknr* in mind, much the same as *lint*. It expects a certain document writing style for \f and \s commands, in that each \fx must be terminated with \fP and each \sx must be terminated with \s0. While it will work to directly go into the next font or explicitly specify the original font or point size, and many existing documents actually do this, such a practice will produce complaints from *checknr*. Since it is probably better to use the \fP and \s0 forms anyway, you should think of this as a contribution to your document preparation style.

SEE ALSO

nroff(1), troff(1), checkeq(1), ms(7), me(7)

DIAGNOSTICS

Complaints about unmatched delimiters.
Complaints about unrecognized commands.
Various complaints about the syntax of commands.

AUTHOR

Mark Horton

BUGS

There is no way to define a 1 character macro name using **−a**.
Does not correctly recognize certain reasonable constructs, such as conditionals.

NAME

chfn – change finger entry

SYNOPSIS

chfn [loginname]

DESCRIPTION

Chfn is used to change information about users. This information is used by the finger program, among others. It consists of the user's "real life" name, office room number, office phone number, and home phone number. *Chfn* prompts the user for each field. Included in the prompt is a default value, which is enclosed between brackets. The default value is accepted simply by typing <return>. To enter a blank field, type the word 'none'. Below is a sample run:

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

Chfn allows phone numbers to be entered with or without hyphens. Because *finger* only knows about UCB extensions, *chfn* will insist upon a four digit number (after the hyphens are removed) for office phone numbers. Also, room numbers must be in Evans or Cory; again, this is because of *finger*.

It is a good idea to run finger after running *chfn* to make sure everything is the way you want it.

The optional argument **loginname** is used to change another person's finger information. This can only be done by the super-user.

FILES

/etc/passwd, /etc/ptmp

SEE ALSO

finger(1), passwd(5)

BUGS

The encoding of the office and extension information is installation dependent.

For historical reasons, the user's name, etc are stored in the passwd file. This is a bad place to store the information. Rumors are that a data base is being developed to store this information, but don't hold your breath.

Because two users may try to write the passwd file at once, a synchronization method was developed. On rare occasions, a message that the password file is "busy" will be printed. In this case, *chfn* sleeps for a while and then tries to write to the passwd file again.

NAME

chgrp − change group

SYNOPSIS

chgrp [-f] group file ...

DESCRIPTION

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

The user invoking *chgrp* must belong to the specified group and be the owner of the file, or be the super-user.

No errors are reported when the **−f** (force) option is given.

FILES

/etc/group

SEE ALSO

chown(2), passwd(5), group(5)

NAME

chmod − change mode

SYNOPSIS

chmod mode file ...

DESCRIPTION

The mode of each named file is 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
2000	set group ID on execution
1000	sticky bit, see *chmod*(2)
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

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*] ...

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for all, or **ugo**. If *who* is omitted, the default is *a* but the setting of the file creation mask (see umask(2)) is taken into account.

Op can be **+** to add *permission* to the file's mode, **−** to take away *permission* and **=** to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group id) and **t** (save text − sticky). Letters **u**, **g** or **o** indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with **=** to take away all permissions.

EXAMPLES

The first example denies write permission to others, the second makes a file executable:

```
chmod o−w file
chmod +x file
```

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter **s** is only useful with **u** or **g**.

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

SEE ALSO

ls(1), chmod(2), stat(2), umask(2), chown(8)

NAME

chsh — change default login shell

SYNOPSIS

chsh name [shell]

DESCRIPTION

Chsh is a command similar to *passwd*(1) except that it is used to change the login shell field of the password file rather than the password entry. If no *shell* is specified then the shell reverts to the default login shell */bin/sh*. Otherwise only */bin/csh*, */bin/oldcsh*, or */usr/new/csh* can be specified as the shell unless you are the super-user.

An example use of this command would be

```
chsh bill /bin/csh
```

SEE ALSO

csh(1), passwd(1), passwd(5)

NAME

ci − check in RCS revisions

SYNOPSIS

ci [options] file ...

DESCRIPTION

Ci stores new revisions into RCS files. Each file name ending in ',v' is taken to be an RCS file, all others are assumed to be working files containing new revisions. *Ci* deposits the contents of each working file into the corresponding RCS file.

Pairs of RCS files and working files may be specified in 3 ways (see also the example section of *co* (1)).

1) Both the RCS file and the working file are given. The RCS file name is of the form *path1/workfile*,v and the working file name is of the form *path2/workfile*, where *path1/* and *path2/* are (possibly different or empty) paths and *workfile* is a file name.

2) Only the RCS file is given. Then the working file is assumed to be in the current directory and its name is derived from the name of the RCS file by removing *path1/* and the suffix ',v'.

3) Only the working file is given. Then the name of the RCS file is derived from the name of the working file by removing *path2/* and appending the suffix ',v'.

If the RCS file is omitted or specified without a path, then *ci* looks for the RCS file first in the directory ./RCS and then in the current directory.

For *ci* to work, the caller's login must be on the access list, except if the access list is empty or the caller is the superuser or the owner of the file. To append a new revision to an existing branch, the tip revision on that branch must be locked by the caller. Otherwise, only a new branch can be created. This restriction is not enforced for the owner of the file, unless locking is set to *strict* (see *rcs* (1)). A lock held by someone else may be broken with the *rcs* command.

Normally, *ci* checks whether the revision to be deposited is different from the preceding one. If it is not different, *ci* either aborts the deposit (if **-q** is given) or asks whether to abort (if **-q** is omitted). A deposit can be forced with the **-f** option.

For each revision deposited, *ci* prompts for a log message. The log message should summarize the change and must be terminated with a line containing a single '.' or a control-D. If several files are checked in, *ci* asks whether to reuse the previous log message. If the std. input is not a terminal, *ci* suppresses the prompt and uses the same log message for all files. See also **-m**.

The number of the deposited revision can be given by any of the options **-r**, **-f**, **-k**, **-l**, **-u**, or **-q** (see **-r**).

If the RCS file does not exist, *ci* creates it and deposits the contents of the working file as the initial revision (default number: 1.1). The access list is initialized to empty. Instead of the log message, *ci* requests descriptive text (see **-t** below).

−r[*rev*] assigns the revision number *rev* to the checked-in revision, releases the corresponding lock, and deletes the working file. This is also the default.

If *rev* is omitted, *ci* derives the new revision number from the caller's last lock. If the caller has locked the tip revision of a branch, the new revision is appended to that branch. The new revision number is obtained by incrementing the tip revision number. If the caller locked a non-tip revision, a new branch is started at that revision by incrementing the highest branch number at that revision. The default initial branch and level numbers are 1. If the caller holds no lock, but he is the owner of the file and locking is not set to *strict*, then the revision is appended to the trunk.

If *rev* indicates a revision number, it must be higher than the latest one on the

branch to which *rev* belongs, or must start a new branch.

If *rev* indicates a branch instead of a revision, the new revision is appended to that branch. The level number is obtained by incrementing the tip revision number of that branch. If *rev* indicates a non-existing branch, that branch is created with the initial revision numbered *rev.1*.

Exception: On the trunk, revisions can be appended to the end, but not inserted.

-f[*rev*] forces a deposit; the new revision is deposited even it is not different from the preceding one.

-k[*rev*] searches the working file for keyword values to determine its revision number, creation date, author, and state (see *co* (1)), and assigns these values to the deposited revision, rather than computing them locally. A revision number given by a command option overrides the number in the working file. This option is useful for software distribution. A revision that is sent to several sites should be checked in with the **-k** option at these sites to preserve its original number, date, author, and state.

-l[*rev*] works like -r, except it performs an additional *co -l* for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is useful for saving a revision although one wants to continue editing it after the checkin.

-u[*rev*] works like -l, except that the deposited revision is not locked. This is useful if one wants to process (e.g., compile) the revision immediately after checkin.

-q[*rev*] quiet mode; diagnostic output is not printed. A revision that is not different from the preceding one is not deposited, unless **-f** is given.

-m*msg* uses the string *msg* as the log message for all revisions checked in.

-n*name* assigns the symbolic name *name* to the number of the checked-in revision. *Ci* prints an error message if *name* is already assigned to another number.

-N*name* same as **-n**, except that it overrides a previous assignment of *name*.

-s*state* sets the state of the checked-in revision to the identifier *state*. The default is *Exp*.

-t[*txtfile*] writes descriptive text into the RCS file (deletes the existing text). If *txtfile* is omitted, *ci* prompts the user for text supplied from the std. input, terminated with a line containing a single '.' or control-D. Otherwise, the descriptive text is copied from the file *txtfile*. During initialization, descriptive text is requested even if -t is not given. The prompt is suppressed if std. input is not a terminal.

DIAGNOSTICS

For each revision, *ci* prints the RCS file, the working file, and the number of both the deposited and the preceding revision. The exit status always refers to the last file checked in, and is 0 if the operation was successful, 1 otherwise.

FILE MODES

An RCS file created by *ci* inherits the read and execute permissions from the working file. If the RCS file exists already, *ci* preserves its read and execute permissions. *Ci* always turns off all write permissions of RCS files.

FILES

The caller of the command must have read/write permission for the directories containing the RCS file and the working file, and read permission for the RCS file itself. A number of temporary files are created. A semaphore file is created in the directory containing the RCS file. *Ci* always creates a new RCS file and unlinks the old one. This strategy makes links to RCS files useless.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.1 ; Release Date: 83/04/04 .

SEE ALSO

co (1), ident(1), rcs (1), rcsdiff (1), rcsintro (1), rcsmerge (1), rlog (1), rcsfile (5), sccstorcs (8).

Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

BUGS

NAME

clear − clear terminal screen

SYNOPSIS

clear

DESCRIPTION

Clear clears your screen if this is possible. It looks in the environment for the terminal type and then in */etc/termcap* to figure out how to clear the screen.

FILES

/etc/termcap terminal capability data base

NAME

cmp − compare two files

SYNOPSIS

cmp [**−l**] [**−s**] file1 file2

DESCRIPTION

The two files are compared. (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 Print the byte number (decimal) and the differing bytes (octal) for each difference.

−s Print nothing for differing files; return codes only.

SEE ALSO

diff(1), comm(1)

DIAGNOSTICS

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

NAME

cnest – check for nested comments in C code

SYNOPSIS

cnest [files ... **]**

DESCRIPTION

Cnest checks for nested comments in C code, since these are not supported by the C compiler and invariably cause obscure problems. It complains about successive "begin comment" fields (/*), end comment fields that do not follow a begin comment field, or comment fields that are not closed before the end of the file. If no input filenames are specified, it reads from the standard input.

SEE ALSO

cc(1)

DIAGNOSTICS

If the program has no nested comments, no diagnostics will be printed. Otherwise, you will see self-explanatory messages giving the line number at which an error was detected.

BUGS

Does not understand about C pre-processor commands, and so can get confused by stuff that is #ifdef'ed out. To get around this, you can use the pipeline

```
cc -E -C filename.c | cnest
```

to have the C pre-processor commands executed but without removing the remaining comments.

AUTHOR

Tom Anderson, John Fluke Mfg. Co., Inc.

NAME

co − check out RCS revisions

SYNOPSIS

co [options] file ...

DESCRIPTION

Co retrieves revisions from RCS files. Each file name ending in ',v' is taken to be an RCS file. All other files are assumed to be working files. *Co* retrieves a revision from each RCS file and stores it into the corresponding working file.

Pairs of RCS files and working files may be specified in 3 ways (see also the example section).

1) Both the RCS file and the working file are given. The RCS file name is of the form *path1/workfile*,v and the working file name is of the form *path2/workfile*, where *path1/* and *path2/* are (possibly different or empty) paths and *workfile* is a file name.

2) Only the RCS file is given. Then the working file is created in the current directory and its name is derived from the name of the RCS file by removing *path1/* and the suffix ',v'.

3) Only the working file is given. Then the name of the RCS file is derived from the name of the working file by removing *path2/* and appending the suffix ',v'.

If the RCS file is omitted or specified without a path, then *co* looks for the RCS file first in the directory ./RCS and then in the current directory.

Revisions of an RCS file may be checked out locked or unlocked. Locking a revision prevents overlapping updates. A revision checked out for reading or processing (e.g., compiling) need not be locked. A revision checked out for editing and later checkin must normally be locked. Locking a revision currently locked by another user fails. (A lock may be broken with the *rcs* (1) command.) *Co* with locking requires the caller to be on the access list of the RCS file, unless he is the owner of the file or the superuser, or the access list is empty. *Co* without locking is not subject to accesslist restrictions.

A revision is selected by number, checkin date/time, author, or state. If none of these options are specified, the latest revision on the trunk is retrieved. When the options are applied in combination, the latest revision that satisfies all of them is retrieved. The options for date/time, author, and state retrieve a revision on the *selected branch.* The selected branch is either derived from the revision number (if given), or is the highest branch on the trunk. A revision number may be attached to one of the options **-l**, **-p**, **-q**, or **-r**.

A *co* command applied to an RCS file with no revisions creates a zero-length file. *Co* always performs keyword substitution (see below).

−l[*rev*] locks the checked out revision for the caller. If omitted, the checked out revision is not locked. See option **-r** for handling of the revision number *rev.*

−p[*rev*] prints the retrieved revision on the std. output rather than storing it in the working file. This option is useful when *co* is part of a pipe.

−q[*rev*] quiet mode; diagnostics are not printed.

−d*date* retrieves the latest revision on the selected branch whose checkin date/time is less than or equal to *date.* The date and time may be given in free format and are converted to local time. Examples of formats for *date*:

22-April-1982, 17:20-CDT,
2:25 AM, Dec. 29, 1983,
Tue-PDT, 1981, 4pm Jul 21 (free format),
Fri, April 16 15:52:25 EST 1982 (output of ctime).

Most fields in the date and time may be defaulted. *Co* determines the defaults in the order year, month, day, hour, minute, and second (most to least significant). At least one of these fields must be provided. For omitted fields that are of higher significance than the highest provided field, the current values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date "20, 10:30" defaults to 10:30:00 of the 20th of the current month and current year. The date/time must be quoted if it contains spaces.

-r[*rev*] retrieves the latest revision whose number is less than or equal to *rev*. If *rev* indicates a branch rather than a revision, the latest revision on that branch is retrieved. *Rev* is composed of one or more numeric or symbolic fields separated by '.'. The numeric equivalent of a symbolic field is specified with the **-n** option of the commands *ci* and *rcs*.

-s*state* retrieves the latest revision on the selected branch whose state is set to *state*.

-w[*login*] retrieves the latest revision on the selected branch which was checked in by the user with login name *login*. If the argument *login* is omitted, the caller's login is assumed.

-j*joinlist* generates a new revision which is the join of the revisions on *joinlist*. *Joinlist* is a comma-separated list of pairs of the form *rev2:rev3*, where *rev2* and *rev3* are (symbolic or numeric) revision numbers. For the initial such pair, *rev1* denotes the revision selected by the options **-l**, ..., **-w**. For all other pairs, *rev1* denotes the revision generated by the previous pair. (Thus, the output of one join becomes the input to the next.)

For each pair, *co* joins revisions *rev1* and *rev3* with respect to *rev2*. This means that all changes that transform *rev2* into *rev1* are applied to a copy of *rev3*. This is particularly useful if *rev1* and *rev3* are the ends of two branches that have *rev2* as a common ancestor. If *rev1* < *rev2* < *rev3* on the same branch, joining generates a new revision which is like *rev3*, but with all changes that lead from *rev1* to *rev2* undone. If changes from *rev2* to *rev1* overlap with changes from *rev2* to *rev3*, *co* prints a warning and includes the overlapping sections, delimited by the lines <<<<<<< *rev1*, =======, and >>>>>>> *rev3*.

For the initial pair, *rev2* may be omitted. The default is the common ancestor. If any of the arguments indicate branches, the latest revisions on those branches are assumed. If the option **-l** is present, the initial *rev1* is locked.

KEYWORD SUBSTITUTION

Strings of the form *$keyword$* and *$keyword:...$* embedded in the text are replaced with strings of the form *$keyword: value $*, where *keyword* and *value* are pairs listed below. Keywords may be embedded in literal strings or comments to identify a revision.

Initially, the user enters strings of the form *$keyword$*. On checkout, *co* replaces these strings with strings of the form *$keyword: value $*. If a revision containing strings of the latter form is checked back in, the value fields will be replaced during the next checkout. Thus, the keyword values are automatically updated on checkout.

Keywords and their corresponding values:

$Author$ The login name of the user who checked in the revision.

$Date$ The date and time the revision was checked in.

$Header$ A standard header containing the RCS file name, the revision number, the date, the author, and the state.

$Locker$ The login name of the user who locked the revision (empty if not locked).

Log The log message supplied during checkin, preceded by a header containing the RCS file name, the revision number, the author, and the date. Existing log messages are NOT replaced. Instead, the new log message is inserted after *$Log:...$*. This is useful for accumulating a complete change log in a source file.

$Revision$ The revision number assigned to the revision.

$Source$ The full pathname of the RCS file.

$State$ The state assigned to the revision with *rcs -s* or *ci -s*.

DIAGNOSTICS

The RCS file name, the working file name, and the revision number retrieved are written to the diagnostic output. The exit status always refers to the last file checked out, and is 0 if the operation was successful, 1 otherwise.

EXAMPLES

Suppose the current directory contains a subdirectory 'RCS' with an RCS file 'io.c,v'. Then all of the following commands retrieve the latest revision from 'RCS/io.c,v' and store it into 'io.c'.

```
co io.c;   co RCS/io.c,v;   co io.c,v;
co io.c RCS/io.c,v;   co io.c io.c,v;
co RCS/io.c,v io.c;   co io.c,v io.c;
```

FILE MODES

The working file inherits the read and execute permissions from the RCS file. In addition, the owner write permission is turned on, unless the file is checked out unlocked and locking is set to *strict* (see *rcs* (1)).

If a file with the name of the working file exists already and has write permission, *co* aborts the checkout if **-q** is given, or asks whether to abort if **-q** is not given. If the existing working file is not writable, it is deleted before the checkout.

FILES

The caller of the command must have write permission in the working directory, read permission for the RCS file, and either read permission (for reading) or read/write permission (for locking) in the directory which contains the RCS file.

A number of temporary files are created. A semaphore file is created in the directory of the RCS file to prevent simultaneous update.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.1 ; Release Date: 83/04/04 .

SEE ALSO

ci (1), ident(1), rcs (1), rcsdiff (1), rcsintro (1), rcsmerge (1), rlog (1), rcsfile (5), sccstorcs (8).

Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

LIMITATIONS

The option **-d** gets confused in some circumstances, and accepts no date before 1970. There is no way to suppress the expansion of keywords, except by writing them differently. In nroff and troff, this is done by embedding the null-character '\&' into the keyword.

BUGS

The option -j does not work for files that contain lines with a single '.'.

NAME

col − filter reverse line feeds

SYNOPSIS

col [−bfx]

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* and output resulting from use of the *tbl*(1) preprocessor.

Although *col* accepts half line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full line boundary. This treatment can be suppressed by the **−f** (fine) option; in this case the output from *col* may contain forward half line feeds (ESC-9), but will still never contain either kind of reverse line motion.

If the **−b** option is given, *col* assumes that the output device in use is not capable of backspacing. In this case, if several characters are to appear in the same place, only the last one read will be taken.

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

Col normally converts white space to tabs to shorten printing time. If the **−x** option is given, this conversion is suppressed.

All control characters are removed from the input except space, backspace, tab, return, newline, 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.

SEE ALSO

troff(1), tbl(1)

BUGS

Can't back up more than 128 lines.
No more than 800 characters, including backspaces, on a line.

NAME

colcrt – filter nroff output for CRT previewing

SYNOPSIS

colcrt [–] [–2] [file ...]

DESCRIPTION

Colcrt provides virtual half-line and reverse line feed sequences for terminals without such capability, and on which overstriking is destructive. Half-line characters and underlining (changed to dashing '–') are placed on new lines in between the normal output lines.

The optional – suppresses all underlining. It is especially useful for previewing *allboxed* tables from *tbl*(1).

The option **–2** causes all half-lines to be printed, effectively double spacing the output. Normally, a minimal space output format is used which will suppress empty lines. The program never suppresses two consecutive empty lines, however. The **–2** option is useful for sending output to the line printer when the output contains superscripts and subscripts which would otherwise be invisible.

A typical use of *colcrt* would be

```
tbl exum2.n | nroff –ms | colcrt – | more
```

SEE ALSO

nroff/troff(1), col(1), more(1), ul(1)

AUTHOR

William Joy

BUGS

Should fold underlines onto blanks even with the '–' option so that a true underline character would show; if we did this, however, *colcrt* wouldn't get rid of *cu'd* underlining completely.

Can't back up more than 102 lines.

General overstriking is lost; as a special case ↑ overstruck with '–' or underline becomes '+'.

Lines are trimmed to 132 characters.

Some provision should be made for processing superscripts and subscripts in documents which are already double-spaced.

NAME

colrm − remove columns from a file

SYNOPSIS

colrm [startcol [endcol]]

DESCRIPTION

Colrm removes selected columns from a file. Input is taken from standard input. Output is sent to standard output.

If called with one parameter the columns of each line will be removed starting with the specified column. If called with two parameters the columns from the first column to the last column will be removed.

Column numbering starts with column 1.

SEE ALSO

expand(1)

AUTHOR

Jeff Schriebman

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, and produces a three column output: lines only in *file1;* lines only in *file2;* and lines in both files. The filename '−' 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** is a no-op.

SEE ALSO

cmp(1), diff(1), uniq(1)

NAME

comp – compose a message

SYNOPSIS

comp [–editor editor] [–form formfile] [file] [–use] [–nouse] [–help]

DESCRIPTION

Comp is used to create a new message to be mailed. If *file* is not specified, the file named "draft" in the user's MH directory will be used. *Comp* copies a message form to the file being composed and then invokes an editor on the file. The default editor is /bin/ned, which may be overridden with the '–editor' switch or with a profile entry "Editor:". The default message form contains the following elements:

```
To:
cc:
Subject:
--------
```

If the file named "components" exists in the user's MH directory, it will be used instead of this form. If '–form formfile' is specified, the specified formfile (from the MH directory) will be used as the skeleton. The line of dashes or a blank line must be left between the header and the body of the message for the message to be identified properly when it is sent (see *send*(1)). The switch '–use' directs *comp* to continue editing an already started message. That is, if a *comp* (or *dist*, *repl*, or *forw*) is terminated without sending the message, the message can be edited again via "comp –use".

If the specified file (or draft) already exists, *comp* will ask if you want to delete it before continuing. A reply of **No** will abort the *comp*, **yes** will replace the existing draft with a blank skeleton, **list** will display the draft, and **use** will use it for further composition.

Upon exiting from the editor, *comp* will ask "What now?". The valid responses are **list**, to list the draft on the terminal; **quit**, to terminate the session and preserve the draft; **quit delete**, to terminate, then delete the draft; **send**, to send the message; **send verbose**, to cause the delivery process to be monitored; **edit <editor>**, to invoke <editor> for further editing; and **edit**, to re-edit using the same editor that was used on the preceding round unless a profile entry "<lasteditor>–next: <editor>" names an alternative editor.

Comp does not affect either the current folder or the current message.

FILES

/etc/mh/components	The message skeleton
or <mh-dir>/components	Rather than the standard skeleton
$HOME/mh_profile	The user profile
<mh-dir>/draft	The default message file
/usr/new/send	To send the composed message

PROFILE COMPONENTS

Path:	To determine the user's MH directory
Editor:	To override the use of /bin/ned as the default editor
<lasteditor>–next:	editor to be used after exit from <lasteditor>

DEFAULTS

'file' defaults to draft

'–editor' defaults to /bin/ned
'–nouse'

NAME

compact, uncompact, ccat — compress and uncompress files, and cat them

SYNOPSIS

compact [name ...]
uncompact [name ...]
ccat [file ...]

DESCRIPTION

Compact compresses the named files using an adaptive Huffman code. If no file names are given, the standard input is compacted to the standard output. *Compact* operates as an on-line algorithm. Each time a byte is read, it is encoded immediately according to the current prefix code. This code is an optimal Huffman code for the set of frequencies seen so far. It is unnecessary to prepend a decoding tree to the compressed file since the encoder and the decoder start in the same state and stay synchronized. Furthermore, *compact* and *uncompact* can operate as filters. In particular,

... | compact | uncompact | ...

operates as a (very slow) no-op.

When an argument *file* is given, it is compacted and the resulting file is placed in *file.C; file* is unlinked. The first two bytes of the compacted file code the fact that the file is compacted. This code is used to prohibit recompaction.

The amount of compression to be expected depends on the type of file being compressed. Typical values of compression are: Text (38%), Pascal Source (43%), C Source (36%) and Binary (19%). These values are the percentages of file bytes reduced.

Uncompact restores the original file from a file compressed by *compact.* If no file names are given, the standard input is uncompacted to the standard output.

Ccat cats the original file from a file compressed by *compact,* without uncompressing the file.

RESTRICTION

The last segment of the filename must contain fewer than thirteen characters to allow space for the appended '.C'.

FILES

*.C compacted file created by compact, removed by uncompact

SEE ALSO

Gallager, Robert G., 'Variations on a Theme of Huffman', *I.E.E.E. Transactions on Information Theory,* vol. IT-24, no. 6, November 1978, pp. 668 - 674.

AUTHOR

Colin L. Mc Master

NAME

congraph – plot connectivity of a graph

SYNOPSIS

congraph [-s*n* -c -n -a] [files ...]

DESCRIPTION

Congraph is a general-purpose utility for plotting the connectivity of graphs. Graphs are described by text files made up of lines describing directed (or non-directed) edges. The format of a line is

<node-1> <node-2>

indicating a connection between the two named nodes. The node-names are taken as ascii text (unless the **–n** option is given, see below) and are separated by spaces or tabs.

The standard output of *congraph* consists of plot commands (see *plot*(5)) which should be piped into *plot*(1G). The plots thus produced have the points given in the input description labeled and connected by straight lines; the points are arranged about a circle in the order first seen in the input file.

For example, here is a three-edge, four-node input file

```
dog cat
cat mouse
house mouse
```

If no **file** arguments are given, the standard input is read. Otherwise, a separate plot is made for each input filename specified. Plots are labeled with the input filename.

OPTIONS

–s*nnn* This sets the scale of the plot produced to *nnn*; the default scale is 1000, so a scale of 500 gives a half-sized plot.

–n This is used when the input node names are all integers, and it is desired to arrange the points on the plot so that they appear in proper numeric order. With this option, if an node numbered between 0 and the highest found in the input file is not on any edge, it still will be plotted (and labeled) in its proper position on the plot.

–c This is used to take an input file where the nodes are described by ascii labels, and convert it into an equivalent description whose nodes are labeled by integers, starting at zero. The nodes are assigned numbers in order of their appearance in the input file. The output description appears on the standard output, and no plot is made. For example, the input file shown above gives the converted description

```
0 1
1 2
3 2
```

–a By default, the graph is considered to be non-directed. This option draws arrowheads to indicate the directions of the edges; the edges are considered to run from the first point on the input line to the second.

SEE ALSO

plot(1)

DIAGNOSTICS

A warning appears if an input file is empty; if too many points or edges are given, the excess ones are silently ignored.

BUGS

The plots produced for large graphs are often too complex for the Dover to print.
On complex plots, the node labels may run together.

NAME

courier – Courier remote procedure call compiler

SYNOPSIS

courier [**-x**] specification

DESCRIPTION

Courier is a compiler for the Mesa-like specification language associated with the Courier remote procedure call protocol.

FILES

Program.cr	Courier specification file for *Program*

The following files are generated by courier from the above:

Program.h	definitions and typedefs
Program_stubs.c	mappings between C and Courier
Program_server.c	server routines
Program_client.c	client routines

SEE ALSO

"Writing Distributed Programs with Courier" by Eric C. Cooper.
"Courier: The Remote Procedure Call Protocol," Xerox System Integration Standard 038112, December 1981.

NAME

cp – copy

SYNOPSIS

cp [**–i**] [**–r**] file1 file2

cp [**–i**] [**–r**] file ... directory

DESCRIPTION

File1 is copied onto *file2*. The mode and owner of *file2* are preserved if it already existed; the mode of the source file is used otherwise.

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

Cp refuses to copy a file onto itself.

If the **–i** option is specified, *cp* will prompt the user with the name of the file whenever the copy will cause an old file to be overwritten. An answer of 'y' will cause *cp* to continue. Any other answer will prevent it from overwriting the file.

If the **–r** option is specified and any of the source files are directories, *cp* copies each subtree rooted at that name; in this case the destination must be a directory.

SEE ALSO

cat(1), pr(1), mv(1)

NAME

cparen - add parentheses to C expressions

SYNOPSIS

cparen [−t *types*]

DESCRIPTION

Written for those of us who can never remember the precedence and associativity of operators in C, **cparen** reads lines of C code from standard-in, adds parentheses to indicate operator binding in expressions, then writes the resultant code to standard-out.

The input code fragment need not contain complete statements. For example, the following line is a valid input to **cparen**:

```
} else if (x->d_prec > 56 && x->d_assoc == LEFT)
```

Normally, **cparen** considers identifiers in expressions to be variable names. The −t option allows you to specify a whitespace-separated list of types. For example,

```
cparen -t 'amap anop'
```

Declares "amap" and "anop" as type names rather than variable names.

DIAGNOSTICS

Exit status is 2 if **cparen** was invoked improperly, 1 if some other error occurred, 0 if all went well.

AUTHOR

Brad Needham, Tektronix, Inc.

BUGS

Cparen assumes that the input code fragment came from a syntactically correct C program -- it does not attempt to give reasonable syntax-error messages.

Because **cparen** focuses on C statments it does not recognize other constructs e.g. variable or function declarations. It cannot process a whole C program.

The input is not filtered through the C preprocessor.

NAME

cref -- cross reference program

SYNOPSIS

cref [- *n*] [-t] file...

DESCRIPTION

Cref generates a complete cross reference listing of one or more C programs, printing the result on the standard output. A listing of the programs with line numbers is printed first, followed by the actual cross reference listing. This latter contains all the programs's symbols alphabetically arranged, one to a line, with each line containing the numbers of the lines in the programs where the symbol was referenced. If the symbol was defined on a given line, that line number will be followed by a '*#*'. Symbols with more than approximately 15 references occupy multiple lines. There is no limit on the number of symbols that *cref* will handle, nor on the number of references per symbol.

Cref stores its symbols in a hash table whose size is determined by *cref* based on the total number of characters in the files to be processed. For almost all programs, this turns out to be an excellent approximation. However, for a few programs, generally short header files, there may be too many symbols for the hash table, and the diagnostic "Hash table overflowed!" will be printed out. Since the output of *cref* is piped through *pr*, it is not really possible for cref to recover from this condition. Instead, *cref* should be rerun with the **-n** option, where **n** is some number. This will multiply the starting size of the hash table by **n** times.

If *cref* is invoked with the -t option, instead of its regular output it produces an output identical in form to that produced by the *ctags(1)* program. The advantage of the *cref* output over *ctags* is that *cref* will flag all variable and macro definitions as well as all function definitions.

AUTHOR

Steve Zimmerman

SEE ALSO

ctags(1)

BUGS

Cref occasionally flags a reference as a definition when it really isn't. This most frequently happens after a **struct**.

NAME

crypt — encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

Crypt reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. *Crypt* encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

will print the clear.

Files encrypted by *crypt* are compatible with those treated by the editor *ed* in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; 'sneak paths' by which keys or cleartext can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e. to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. No doubt the choice of keys and key security are the most vulnerable aspect of *crypt*.

FILES

/dev/tty for typed key

SEE ALSO

ed(1), makekey(8)

BUGS

There is no warranty of merchantability nor any warranty of fitness for a particular purpose nor any other warranty, either express or implied, as to the accuracy of the enclosed materials or as to their suitability for any particular purpose. Accordingly, Bell Telephone Laboratories assumes no responsibility for their use by the recipient. Further, Bell Laboratories assumes no obligation to furnish any assistance of any kind whatsoever, or to furnish any additional information or documentation.

NAME

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

SYNOPSIS

csh [−cefinstvVxX] [arg ... **]**

DESCRIPTION

Csh is a first implementation of a command language interpreter incorporating a history mechanism (see **History Substitutions**) job control facilities (see **Jobs**) and a C-like syntax. So as to be able to use its job control facilities, users of *csh* must (and automatically) use the new tty driver fully described in *tty*(4). This new tty driver allows generation of interrupt characters from the keyboard to tell jobs to stop. See *stty*(1) for details on setting options in the new tty driver.

An instance of *csh* begins by executing commands from the file '.cshrc' in the *home* directory of the invoker. If this is a login shell then it also executes commands from the file '.login' there. It is typical for users on crt's to put the command "stty crt" in their *.login* file, and to also invoke *tset*(1) there.

In the normal case, the shell will then begin reading commands from the terminal, prompting with '% '. Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and then parsed. Finally each command in the current line is executed.

When a login shell terminates it executes commands from the file '.logout' in the users home directory.

Lexical structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters '&' '|' ';' '<' '>' '(' ')' form separate words. If doubled in '&&', '||', '<<' or '>>' these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with '\'. A newline preceded by a '\' is equivalent to a blank.

In addition strings enclosed in matched pairs of quotations, '`', '' or '"', form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of '' or '"' characters a newline preceded by a '\' gives a true newline character.

When the shell's input is not a terminal, the character '#' introduces a comment which continues to the end of the input line. It is prevented this special meaning when preceded by '\' and 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 ';', and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an '&'.

Any of the above may be placed in '(' ')' to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with '||' or '&&' indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See *Expressions.)*

Jobs

The 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 jobs which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit the key ^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. There is another special key ^Y which does not generate a STOP signal until a program attempts to *read*(2) it. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them.

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'. Just naming a job brings it to the foreground; thus '%1' is a synonym for 'fg %1', bringing job 1 back into the foreground. Similarly saying '%1 &' resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them, if these prefixes are unambiguous, thus '%ex' would normally restart a suspended *ex*(1) job, if there were only one suspended job whose name began with the string 'ex'. It is also possible to say '%?string' which specifies a job whose text contains *string*, if there is only one such job.

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. For close analogy with the syntax of the *history* mechanism (described below), '%%' is also a synonym for the current job.

Status reporting

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. If, however, you set the shell variable *notify*, the shell will notify you immediately of changes of status in background jobs. There is also a shell command *notify* which marks a single process so that its status changes will be immediately reported. By default *notify* marks the current process; simply say 'notify' after starting a background job to mark it.

When you try to leave the shell while jobs are stopped, you will be warned that 'You have stopped 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 suspended jobs will be terminated.

Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur.

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 the previous command with little typing and a high degree of confidence. History substitutions begin with the character '!' and may begin **anywhere** in the input stream (with the proviso that they **do not** nest.) This '!' may be preceded by an '\' 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 '↑'. This special abbreviation will be described later.) Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

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

For definiteness, 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 the current event number can be made part of the *prompt* by placing an '!' in the prompt string.

With the current event 13 we can refer to previous events by event number '!11', relatively as in '!−2' (referring to the same event), 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 '!!' refers to the previous command; thus '!!' alone is essentially a *redo*.

To select words from an event we can follow the event specification by a ':' and a designator for the desired words. The words of a input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. 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 of the editors, but rather strings. Any character may be used as the delimiter in place of '/'; a '\' quotes the delimiter into the *l* and *r* strings. The character '&' in the right hand side is replaced by the text from the left. A '\' quotes '&' also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in '!?*s*?'. The trailing delimiter in the substitution may be omitted if a newline follows immediately as may the trailing '?' in a contextual scan.

A history reference may be given without an event specification, e.g. '!$'. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus '!?foo?↑ !$' 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↑' providing a convenient shorthand for substitutions on the text of the previous line. Thus '↑lb↑lib' fixes the spelling of 'lib' in the previous command. Finally, a history substitution may be surrounded with '{' and '}' if necessary to insulate it from the characters which follow. Thus, after 'ls −ld ~paul' we might do '!{l}a' to do 'ls −ld ~paula', while '!la' would look for a command starting 'la'.

Quotations with ' and "

The quotation of strings by "'" and '"' can be used to prevent all or some of the remaining substitutions. Strings enclosed in "'" are prevented any further interpretation. Strings enclosed in '"' may be expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see *Command Substitition* below) does a '"' quoted string yield parts of more than one word; "'" quoted strings never do.

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread 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, then the argument list is left unchanged.

Thus if the alias for 'ls' is 'ls −l' the command 'ls /usr' would map to 'ls −l /usr', the argument list here being undisturbed. Similarly if the alias for 'lookup' was 'grep !↑ /etc/passwd' then 'lookup bill' would map to 'grep bill /etc/passwd'.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

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

Variable substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the *set* and *unset* commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The setting of this variable results from the −v command line option.

Other operations treat variables numerically. The '@' command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by '$' characters. This expansion can be prevented by preceding the '$' with a '\' except within '"'s where it **always** occurs, and within "'"s where it **never** occurs. Strings quoted by "`" are interpreted later (see *Command substitution* below) so '$' substitution does not occur there until later, if at all. A '$' 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. It is thus possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in '"' or given the ':q' modifier the results of variable substitution may eventually be command and filename substituted. Within '"' a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variables value separated by blanks. When the ':q' modifier is applied to a substitution the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$name
${name}

Are replaced 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, then that value is returned

(but : modifiers and the other forms given below are not available in this case).

$name[selector]
${name[selector]}
May be used to select only some of the words from the value of *name*. The selector is subjected to '$' substitution and may consist of a single number or two numbers separated by a '−'. The first word of a variables value is numbered '1'. If the first number of a range is omitted it defaults to '1'. If the last member of a range is omitted it defaults to '$#name'. The selector '*' selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$#name
${#name}
Gives the number of words in the variable. This is useful for later use in a '[selector]'.

$0
Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$number
${number}
Equivalent to '$argv[number]'.

$*
Equivalent to '$argv[*]'.

The modifiers ':h', ':t', ':r', ':q' and ':x' may be applied to the substitutions above as may ':gh', ':gt' and ':gr'. If braces '{' '}' appear in the command form then the modifiers must appear within the braces. **The current implementation allows only one ':' modifier on each '$' expansion.**

The following substitutions may not be modified with ':' modifiers.

$?name
${?name}
Substitutes the string '1' if name is set, '0' if it is not.

$?0
Substitutes '1' if the current input filename is known, '0' if it is not.

$$
Substitute the (decimal) process number of the (parent) shell.

$<
Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is 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

Command substitution is indicated by a command enclosed in `. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within `"'s, 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 '{' or begins with the character '~', then that word is a candidate for filename substitution, also known as 'globbing'. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the meta-characters '*', '?' and '[' imply pattern matching, the characters '~' and '{' being more akin to abbreviations.

In matching filenames, the character '.' at the beginning of a filename or immediately following a '/', as well as the character '/' must be matched explicitly. The character '*' matches any string of characters, including the null string. The character '?' matches any single character. The sequence '[...]' matches any one of the characters enclosed. Within '[...]', a pair of characters separated by '−' matches any character lexically between the two.

The character '~' at the beginning of a filename is used to refer to home directories. Standing alone, i.e. '~' it expands to the invokers home directory as reflected in the value of the variable *home*. When followed by a name consisting of letters, digits and '−' characters the shell searches for a user with that name and substitutes their home directory; thus '~ken' might expand to '/usr/ken' and '~ken/chmach' to '/usr/ken/chmach'. If the character '~' is followed by a character other than a letter or '/' or appears not at the beginning of a word, it is left undisturbed.

The metanotation 'a{b,c,d}e' is a shorthand for 'abe ace ade'. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. 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 '{', '}' and '{}' are passed undisturbed.

Input/output

The standard input and standard output of a command may be redirected with the following syntax:

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

<< word
: Read the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting '\', '"', '' or '`' appears in *word* variable and command substitution is performed on the intervening lines, allowing '\' to quote '$', '\' and '`'. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name
>! name
>& name

>&! name

The file *name* is used as standard output. If the file does not exist then it is created; if the file exists, its is truncated, its previous contents being lost.

If the variable *noclobber* is set, then the file must not exist or be a character special file (e.g. a terminal or '/dev/null') or an error results. This helps prevent accidental destruction of files. In this case the '!' forms can be used and suppress this check.

The forms involving '&' route the diagnostic output into the specified file as well as the standard output. *Name* is expanded in the same way as '<' input filenames are.

>> name
>>& name
>>! name
>>&! name

Uses file *name* as standard output like '>' but places output at the end of the file. If the variable *noclobber* is set, then it is an error for the file not to exist unless one of the '!' forms is given. Otherwise similar to '>'.

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 inline 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, then the process will block and the user will be notified (see **Jobs** above.)

Diagnostic output may be directed through a pipe with the standard output. Simply use the form '|&' rather than just '|'.

Expressions

A number of the builtin commands (to be described subsequently) take expressions, in which the operators are similar to those of C, 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, '==' '!=' '=~' and '!~', '<=' '>=' '<' and '>', '<<' and '>>', '+' and '-', '*' '/' and '%' being, in groups, at the same level. The '==' '!=' '=~' and '!~' 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, e.g. '*'s, '?'s and instances of '[...]') against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings which begin with '0' are considered octal numbers. Null or missing arguments are considered '0'. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser ('&' '|' '<' '>' '(' ')') they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in '{' and '}' and file enquiries of the form '-*l* name' where *l* is one of:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero size
f	plain file
d	directory

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries return false, i.e. '0'. Command executions succeed, returning true, i.e. '1', if the command exits with status 0, otherwise they fail, returning false, i.e. '0'. If more detailed status information is required then the command should be executed outside of an expression and the variable *status* examined.

Control flow

The shell contains a number of commands which can be used 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 up 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 will succeed on non-seekable inputs.)

Builtin commands

Builtin commands are executed within the shell. If a builtin command occurs as any component of a pipeline except the last then it is executed in a subshell.

alias
alias name
alias name wordlist
The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name; wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias.*

alloc
Shows the amount of dynamic core in use, broken down into used and free core, and address of the last location in the heap. With an argument shows each used and free block on the internal dynamic memory chain indicating its address, size, and whether it is used or free. This is a debugging command and may not work in production versions of the shell; it requires a modified version of the system memory allocator.

bg
bg %job ...
Puts the current or specified jobs into the background, continuing them if they were stopped.

break
Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while.* The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw
Causes a break from a *switch*, resuming after the *endsw*.

case label:
A label in a *switch* statement as discussed below.

cd
cd name
chdir
chdir name
Change the shells working directory to directory *name*. If no argument is given then 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 '../'), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with '/', then this is tried to see if it is a directory.

continue
Continue execution of the nearest enclosing *while* or *foreach*. The rest of the commands on the current line are executed.

default:
Labels the default case in a *switch* statement. The default should come after all *case* labels.

dirs
Prints the directory stack; the top of the stack is at the left, the first directory in the stack being the current directory.

echo wordlist
echo **−n** wordlist
The specified words are written to the shells standard output, separated by spaces, and terminated with a newline unless the **−n** option is specified.

else
end
endif
endsw
See the description of the *foreach*, *if*, *switch*, and *while* statements below.

eval arg ...
(As in *sh*(1).) The arguments are read as input to the shell and the resulting command(s) executed in the context of the current shell. This is usually 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
The specified command is executed in place of the current shell.

exit
exit(expr)
The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

fg
fg %job ...
Brings the current or specified jobs into the foreground, continuing them if they were stopped.

foreach name (wordlist)
...
end

The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching *end* are executed. (Both *foreach* and *end* must appear alone on separate lines.)

The builtin command *continue* may be used to continue the loop prematurely and the builtin command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with '?' before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

glob wordlist

Like *echo* but no '\' escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

goto word

The specified *word* is filename and command expanded to yield a string of the form 'label'. The shell rewinds its input as much as possible and searches for a line of the form 'label:' possibly preceded by blanks or tabs. Execution continues after the specified line.

hashstat

Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding *exec*'s). An *exec* is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component which does not begin with a '/'.

history
history *n*
history **-r** *n*
history **-h** *n*

Displays the history event list; if *n* is given only the *n* most recent events are printed. The **-r** option reverses the order of printout to be most recent first rather than oldest first. The **-h** option causes the history list to be printed without leading numbers. This is used to produce files suitable for sourceing using the -h option to *source*.

if (expr) command

If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *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, when command is **not** executed (this is a bug).

if (expr) **then**
...
else if (expr2) **then**
...
else
...
endif

If the specified *expr* is true then the commands to the first *else* are executed; else if *expr2* is true then the commands to the second else are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise 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.)*

jobs
jobs −l
Lists the active jobs; given the −l options lists process id's in addition to the normal information.

kill %job
kill −sig %job ...
kill pid
kill −sig pid ...
kill −l
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 names are listed by "kill −l". There is no default, saying just 'kill' does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal as well.

limit
limit *resource*
limit *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 no *maximum-use* is given, then the current limit is printed; if no *resource* is given, then all limitations are given.

Resources controllable currently include *cputime* (the maximum number of cpu-seconds to be used by each process), *filesize* (the largest single file which can be created), *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).

The *maximum-use* may be given 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); a scale factor of 'm' or 'megabytes' may also be used. For *cputime* the default scaling is 'seconds', while 'm' for minutes or 'h' for hours, or a time of the form 'mm:ss' giving minutes and seconds may be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

login
Terminate a login shell, replacing it with an instance of **/bin/login.** This is one way to log off, included for compatibility with *sh*(1).

logout
Terminate a login shell. Especially useful if *ignoreeof* is set.

nice
nice +number
nice command
nice +number command
The first form sets the *nice* for this shell to 4. The second form sets the *nice* to the given number. The final two forms run command at priority 4 and *number* respectively. The super-user may specify negative niceness by using 'nice −number ...'. Command is always executed in a sub-shell, and the restrictions place on commands in simple *if* statements apply.

nohup

nohup command
The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with '&' are effectively *nohup'ed.*

notify
notify %job ...
Causes the shell to 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
onintr −
onintr label
Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form 'onintr −' causes all interrupts to be ignored. The final form causes the shell to execute a 'goto label' when an interrupt is received or a child process terminates because it was interrupted.

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.

popd
popd +n
Pops the directory stack, returning to the new top directory. With a argument '+*n*' discards the *n*th entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

pushd
pushd name
pushd +n
With no arguments, *pushd* exchanges the top two elements of the directory stack. Given a *name* argument, *pushd* changes to the new directory (ala *cd)* and pushes the old current working directory (as in *csw)* onto the directory stack. With a numeric argument, rotates the *n*th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

rehash
Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command
The specified *command* which is subject to the same restrictions as the *command* in the one line *if* statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

set
set name
set name=word
set name[index]=word
set name=(wordlist)
The first form of the command shows the value of all shell variables. Variables which

have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *index'th* component of name to word; this component must already exist. The final form sets *name* to the list of words in *wordlist*. In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

setenv name value
Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variable USER, TERM, and PATH are automatically imported to and exported from the *csh* variables *user*, *term*, and *path*; there is no need to use *setenv* for these.

shift
shift variable
The members of *argv* are shifted to the left, discarding *argv[1]*. It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

source name
source **−h** name
The shell reads commands from *name*. *Source* commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands. Normally input during *source* commands is not placed on the history list; the −h option causes the commands to be placed in the history list without being executed.

stop
stop %job ...
Stops the current or specified job which 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 ^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
time command
With no argument, a summary of time used by this shell and its children is printed. 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
umask value
The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern
All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by 'unalias *'. It is not an error for nothing to be *unaliased.*

unhash
Use of the internal hash table to speed location of executed programs is disabled.

unlimit *resource*
unlimit
Removes the limitation on *resource.* If no *resource* is specified, then all *resource* limitations are removed.

unset pattern
All variables whose names match the specified pattern are removed. Thus all variables are removed by 'unset *'; this has noticeably distasteful side-effects. It is not an error for nothing to be *unset.*

unsetenv pattern
Removes all variables whose name match the specified pattern from the environment. See also the *setenv* command above and *printenv*(1).

wait
All background jobs are waited for. It the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (expr)
...
end
While the specified expression evaluates non-zero, the commands between the *while* and the matching end are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. (The *while* and *end* must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

%job
Brings the specified job into the foreground.

%job &
Continues the specified job in the background.

@
@ name = expr
@ name[index] = expr
The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr.* If the expression contains '<', '>', '&' or '|' then at least this part of the expression must be placed within '(' ')'. The third form assigns 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. Spaces are, however, mandatory in separating components of *expr* which would otherwise be single words.

Special postfix '++' and '--' operators increment and decrement *name* respectively, i.e. '@ i++'.

Pre-defined and environment variables

The following variables have special meaning to the shell. Of these, *argv, cwd, home, path, prompt, shell* and *status* are always set by the shell. Except for *cwd* and *status* this setting occurs only at initialization; these variables will not then be modified unless this is done explicitly by the user.

This shell copies the environment variable USER into the variable *user*, TERM into *term*, and HOME into *home*, and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file *.cshrc* as inferior *csh* processes will 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, i.e. '$1' is replaced by '$argv[1]', etc.

cdpath Gives a list of alternate directories searched to find subdirectories in *chdir* commands.

cwd The full pathname of the current directory.

echo Set when the -x command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively.

histchars Can be given a string value to change the characters used in history substitution. The first character of its value is used as the history substitution character, replacing the default character !. The second character of its value replaces the character ↑ in quick substitutions.

history Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. Too large values of *history* may run the shell out of memory. The last executed command is always saved on the history list.

home The home directory of the invoker, initialized from the environment. The filename expansion of '~' refers to this variable.

ignoreeof If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-D's.

mail The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says '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, which is 10 minutes.

If multiple mail files are specified, then the shell says 'New mail in *name*' when there is mail in the file *name*.

noclobber As described in the section on *Input/output*, restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that '>>' redirections refer to existing files.

noglob If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e. 'echo [' still gives an error.

notify If set, the shell notifies asynchronously of job completions. The default is to rather present 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 then only full path names will execute. The usual search path is '.', '/bin' and '/usr/bin', but this may vary from system to system. For the super-user the default search path is '/etc', '/bin' and '/usr/bin'. A shell which is given neither the **-c** nor the **-t** option will normally hash 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* or the commands may not be found.

prompt The string which is printed before each command is read from an interactive terminal input. If a '!' appears in the string it will be replaced by the current event number unless a preceding '\' is given. Default is '% ', or '# ' for the super-user.

savehist is given a numeric value to control the number of entries of the history list that are saved in ~/.history when the user logs out. Any command which has been referenced in this many events will be saved. During start up the shell sources ~/.history into the history list enabling history to be saved across logins. Too large values of *savehist* will slow down the shell during start up.

shell The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of *Non-builtin Command Execution* below.) Initialized to the (system-dependent) home of the shell.

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status '1', all other builtin commands set status '0'.

time Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.

verbose Set by the **-v** command line option, causes the words of each command to be printed after history substitution.

Non-builtin command execution

When a command to be executed is found to not be a builtin command the shell attempts to execute the command via *execve*(2). Each word in the variable *path* names a directory from which the shell will attempt to execute the command. If it is given neither a **-c** nor a **-t** option, the shell will hash the names in these directories into an internal table so that it will

only try an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash*), or if the shell was given a **-c** or **-t** argument, and in any case for each directory component of *path* which does not begin with a '/', the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus '(cd ; pwd) ; pwd' prints the *home* directory; leaving you where you were (printing this after the home directory), while 'cd ; pwd' leaves you in the *home* directory. Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an *alias* for *shell* then the words of the alias will be prepended to the argument list to form the shell command. The first word of the *alias* should be the full path name of the shell (e.g. '$shell'). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

Argument list processing

If argument 0 to the shell is '-' then this is a login shell. The flag arguments are interpreted as follows:

-c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.

-e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

-f The shell will start faster, because it will neither search for nor execute commands from the file '.cshrc' in the invokers home directory.

-i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

-n Commands are parsed, but not executed. This aids in syntactic checking of shell scripts.

-s Command input is taken from the standard input.

-t A single line of input is read and executed. A '\' may be used to escape the newline at the end of this line and continue onto another line.

-v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.

-x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.

-V Causes the *verbose* variable to be set even before '.cshrc' is executed.

-X Is to **-x** as **-V** is to **-v**.

After processing of flag arguments if arguments remain but none of the **-c**, **-i**, **-s**, or **-t** options was given 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, the shell will execute such a 'standard' shell if the first character of a script is not a '#', i.e. if the script does not start with a comment. Remaining arguments initialize the variable *argv*.

Signal handling

The shell 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 which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *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 login shell is reading the file '.logout'.

AUTHOR

William Joy. Job control and directory stack features first implemented by J.E. Kulp of I.I.A.S.A, Laxenburg, Austria, with different syntax than that used now.

FILES

~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by login shell, after '.cshrc' at login.
~/.logout	Read by login 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'.

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves filename expansion is limited to 1/6'th 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.

SEE ALSO

sh(1), access(2), execve(2), fork(2), killpg(2), pipe(2), sigvec(2), umask(2), setrlimit(2), wait(2), tty(4), a.out(5), environ(7), 'An introduction to the C shell'

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 (i.e. wrong) as the job may have changed directories internally.

Shell builtin 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 will then immediately execute 'c'. This is especially noticeable if this expansion results from an *alias*. It suffices to place the sequence of commands in ()'s to force it to a subshell, i.e. '(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; shell procedures should be provided 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.

Symbolic links fool the shell. In particular, *dirs* and 'cd ..' don't work properly once you've crossed through a symbolic link.

NAME

ctags − create a tags file

SYNOPSIS

ctags [**−BFatuwvx**] name ...

DESCRIPTION

Ctags makes a tags file for *ex*(1) from the specified C, Pascal and Fortran sources. A tags file gives the locations of specified objects (in this case functions and typedefs) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern, typedefs with a line number. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the *tags* file, *ex* can quickly find these objects definitions.

If the **−x** flag is given, *ctags* produces a list of object names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

If the **−v** flag is given, an index of the form expected by *vgrind*(1) is produced on the standard output. This listing contains the function name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through **sort −f**. Sample use:

```
ctags −v files | sort −f > index
vgrind −x index
```

Files whose name ends in **.c** or **.h** are assumed to be C source files and are searched for C routine and macro definitions. Others are first examined to see if they contain any Pascal or Fortran routine definitions; if not, they are processed again looking for C definitions.

Other options are:

−F use forward searching patterns (/.../) (default).

−B use backward searching patterns (?...?).

−a append to tags file.

−t create tags for typedefs.

−w suppressing warning diagnostics.

−u causing the specified files to be *updated* in tags, that is, all references to them are deleted, and the new values are appended to the file. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.)

The tag *main* is treated specially in C programs. The tag formed is created by prepending *M* to the name of the file, with a trailing .c removed, if any, and leading pathname components also removed. This makes use of *ctags* practical in directories with more than one program.

FILES

tags	output tags file

SEE ALSO

ex(1), vi(1)

AUTHOR

Ken Arnold; FORTRAN added by Jim Kleckner; Bill Joy added Pascal and **−x**, replacing *cxref*; C typedefs added by Ed Pelegri-Llopart.

BUGS

Recognition of **functions**, **subroutines** and **procedures** for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name you lose.

The method of deciding whether to look for C or Pascal and FORTRAN functions is a hack.

Does not know about #ifdefs.

Should know about Pascal types. Relies on the input being well formed to detect typedefs. Use of -tx shows only the last line of typedefs.

NAME

cxref – cross reference C source files

SYNOPSIS

cxref [−SCcfis] [−w *width*] [files]

DESCRIPTION

Cxref reads the named C source files and produces on the standard output a cross reference of all the identifiers and constants in the files. Constants are integer constants (12, 0421, 0x1A), floating point constants (123.45, 0.2e-4), string constants ("this is a string\n"), and character constants ('a', '\033'). Identifiers, character constants, and string constants are sorted lexicographically, i.e. according to the machine collating sequence (7-bit ASCII on the Vax or Pyramid). Integer and floating point constants are sorted numerically. The trailing 'l' or 'L' on long integer constants will not show up in the output listing.

If no files are named, *cxref* reads the standard input. For multiple files, the argument "−" (a single dash) indicates that the standard input should be read at that point.

If arguments are given, they must come before any file names.

Cxref recognizes the following arguments:

−S Cross reference all files separately. The default action is to cross reference all named files together.

−c Leave character constants out of the cross reference listing.

−f Leave floating point constants out of the cross reference listing.

−i Leave integer constants out of the cross reference listing.

−s Leave string constants out of the cross reference listing.

−C Leave *all* constants, character, string, integer, and floating point, out of the cross reference listing. By default, all types of constants are included in the cross reference.

−w *width*
Make the output be *width* columns wide. The output width will never be less than 51 or more than 132 columns. *Cxref* silently adjusts incorrect settings to the nearest allowable setting. If no width is specified, the output will default to 80 columns wide.

Cxref does *not* include #include files, or expand macro definitions. Files named in #include lines can be listed on the command line if they should also be cross referenced.

If a quoted string has an escaped newline in it (see "The C Programming Language", page 181, or Section 2.5 of the C Reference Manual), it will show up inside the string in the output listing as \N. This is to make it visible to the programmer, and to keep the various filters which *Cxref* uses to actually do the work from getting terribly confused.

Cxref is best run in the background, with its output redirected into a file or the line printer spooler *lpr*(1), since it reads all the named files, using *sort*(1) as an intermediate pass. The sorting can take time which the user can probably put to more productive use.

DIAGNOSTICS

Self explanatory.

BUGS

Systems running UNIX 4.0 and later already have a program named *cxref*. Therefore, on those systems, this program should be renamed.

Cxref does not do any formatting on its output (other than to insure that it writes the proper number of columns), so it should probably be run piping its output into *pr*(1).

Floating point constants are converted to a common format for sorting, therefore they may appear in the output in a format different from (but numerically equivalent to) their form in the original source code.

SEE ALSO

lex(1), *lpr*(1), *pr*(1), *sort*(1)

FILES

/tmp/cxr.$$.*

temporary files for integer and floating point contstants. *Cxref* removes these files when it is through.

AUTHOR

Arnold Robbins
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Geogia 30332

UUCP: gatech!arnold
CSNET: arnold@gatech
ARPANET: arnold%gatech.csnet@csnet-relay.arpa

NAME

cz (czarina) – convert files to press format and print them on the Dover.

SYNOPSIS

cz [options] [files]

DESCRIPTION

cz (czarina -- the VAX version of the Alto program empress) reads in text files, converts them to press format and ships them to the Dover.

The environment variable CZ may be used to specify defaults. The value of CZ is parsed as a string of arguments before the arguments that appear on the command line. For example "CZ='-f TimesRoman8'" sets your default body font to 8 point Times Roman.

If a file is already in press format czarina will discover this fact by examination. Such files will be shipped as-is with minor changes to the document directory to make the cover sheet of the final output agree with the selected options.

A font name has three parts: A family name (from the set TimesRoman, Helvetica, Gacha, and Sail; Gacha and Sail are fixed width), an optional point size (1 point=1/72 inch -- 8 point is a good small font), and an optional facing (b=bold, i=italic, nothing=roman). So Gacha8b is 8 point bold Gacha, Helvetica12i is 12 point italic Helvetica.

The possible options are:

-2	Sets two column mode.
-b banner	Uses "banner" to label the output. It will appear on the cover page on the line labeled "File:".
-c n	Causes n copies of the output to be produced. The default is one.
-f font	Sets the font to be used for the body of each page. Defaults to Gacha10, unless two column rotated mode is used, in which case it defaults to Gacha8.
-F font	Sets the font to be used for page headings. Defaults to Helvetica12
-g	Causes the fact that a file is garbage to be ignored. Normally, any file with more than a small number of non-printing characters is suspected of being garbage, and not printed -- this option means "print it anyway."
-h header	Sets the string to be used for page headings to *header*. The default header is constructed from the file name, its last modification date, and a page number.
-l	Causes line printer simulation mode to be used: pages will be 66 lines long and headers will be omitted.
-n name	Causes the given name to be used as the delivery address of your output (the "For: " field on the cover sheet). This usually defaults to your full name, causing output to be filed by the first character of your first name. If you'd rather go by your last name, then use '-n "Bovik, Harold"'. You can arrange to have this done by default by using the CZ environment variable.
-N	Causes the Dover printer daemon to notify you when the job is done, by writing to your terminal.
-o	If *cz* cannot find characters in a font, the missing characters are listed.

-p name — Causes the press file to be written to the named file rather than being shipped to the Dover.

-r — Causes the output to be rotated 90 degrees on the page (landscape mode). This is good for output that requires a wide page or for program listings when used in conjunction with two column mode.

"cz -2r files" is the 'approved' way to get program listings on the Dover.

-s pagespec — Allows you to specify a range (or several ranges) of pages to be printed. *Pagespec* is a string, not containing spaces, of the form "pagerange[,pagespec]". *Pagerange* is either a single page number, or a range of the form "lo:hi" or "lo-hi". The characters '$' and '*' both stand for the last page. (If you use '-' in a range, the normal feedback of page ranges onto stdout will be silenced.)

-S — Causes the press file created to be written in Alto byte order instead of PDP-11 byte order. Note that the Dover printer daemon doesn't care about which order your file is in; it can handle either order.

-t — Causes page titles to be omitted.

ENVIRONMENT

CZ string of options to be used by *cz*.

FILES

/usr/stanford/lib/fonts.widths describes all the available fonts.
/usr/stanford/bin/dpr Dover Printer spooler

SEE ALSO

dpr (1), pr (1), dpq (1), dprm (1), dumpfonts(1)

DIAGNOSTICS

Lots, but they should be self explanatory.

BUGS

The fonts available on the Dover do not necessarily correspond to the the descriptions in the "fonts.widths" data base, hence you may ask for a font, czarina will be happy, but the Dover will substitute something else.

If you give the -p argument after the file to be converted to press format, cz will lie and tell you that the file has been put in the right place, when in fact it is in some obscure temporary file. Always give your -p argument first.

The name of the -N option is too close to the -n option. Moreover, if you are logged in on more than one terminal when your job finished, the notification will be written to one of them, chosen (seemingly) at random.

NAME

dataio – load the data i/o prom programmer

SYNOPSIS

dataio [format-code] datafile

DESCRIPTION

Dataio is a shell script which provides a helping hand in loading *datafile* into the data i/o prom programmer. It sets the format code for data transfer, loads *datafile* into the programmer, and gives helpful messages. If **[format-code]** is not specified, 83 (Intel hex) is assumed.

AUTHOR

Erik Hedberg

SEE ALSO

The manual near the programmer.

BUGS

No checking of the format code is done. If it is more than two characters, the first two will be used as the format code and the dataio will try to interpret the rest as commands.

NAME

date − print and set the date

SYNOPSIS

date [-u] [yymmddhhmm [.ss]]

DESCRIPTION

If no arguments are given, the current date and time are printed. If a date is specified, the current date is set. The *-u* flag is used to display the date in GMT (universal) time. This flag may also be used to set GMT time. *yy* is the last two digits of the year; 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; *.ss* is optional and is the seconds. For example:

date 10080045

sets the date to Oct 8, 12:45 AM. The year, month and day may be omitted, the current values being the defaults. The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight time.

FILES

/usr/adm/wtmp to record time-setting

SEE ALSO

utmp(5)

DIAGNOSTICS

'Failed to set date: Not owner' if you try to change the date but are not the super-user.

BUGS

The system attempts to keep the date in a format closely compatible with VMS. VMS, however, uses local time (rather than GMT) and does not understand daylight savings time. Thus if you use both UNIX and VMS, VMS will be running on GMT.

NAME

dbadd -- add entry to an Emacs data base
dbcreate -- create an Emacs data base
dblist -- list contents of an Emacs data base
dbprint -- print an entry from an Emacs data base

SYNOPSIS

dbadd dbname key
dbcreate dbname
dblist dbname **[-l] [-p** newdbname **]**
dbprint dbname key

DESCRIPTION

All of these commands deal with databases used by the Unix Emacs database manipulation facilities. A Unix Emacs database is simply a set of (key,content) pairs, as in dbm(3), except that the content part can be a very long string.

Dbadd adds the text from the standard input to the named database using the given key. **Dbcreate** creates the named database, making it empty. **Dbprint** prints the contents of the entry from the database with the given key.

Dblist with no arguments simply lists the keys of all he items in the database. With the -l option it prints some internal information from the database of no interest to anyone but the implementor. The **-p** option causes the key and content of every entry to be listed as a shell command file which when executed will repeatedly invoke **dbadd** to rebuild the database. This form of **dblist** is handy when you want a readable ascii file representation of a data base for shipping around or editing. Databases should be recreated periodically to garbage collect them.

FILES

dbname.dir, *dbname*.pag, and *dbname*.dat: the three component subfiles of a database.

SEE ALSO

James Gosling, *The Unix Emacs Manual*
emacs(1), dbm (3) from which much code was stolen.

AUTHOR

James Gosling @ CMU

NAME

dbx – debugger

SYNOPSIS

dbx [**–r**] [**–i**] [**–I** *dir*] [*objfile* [*coredump*]]

DESCRIPTION

Dbx is a tool for source level debugging and execution of programs under UNIX. 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. Currently, *cc*(1), *pc*(1), *f77*(1), and the DEC Western Research Laboratory Modula-2 compiler, *mod*(1), produce the appropriate source information. 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 a file named "core" exists in the current directory or a *coredump* file is specified, *dbx* can be used to examine the state of the program when it faulted.

If the file ".dbxinit" exists in the current directory then the debugger commands in it are executed. *Dbx* also checks for a ".dbxinit" in the user's home directory if there isn't one in the current directory.

The command line options and their meanings are:

–r Execute *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* will read from "/dev/tty" when **–r** is specified and standard input is not a terminal.

–i Force *dbx* to act as though standard input is a terminal.

–I *dir* Add *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.

Unless **–r** is specified, *dbx* just prompts and waits for a command.

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*
ignore *number*
Start or stop trapping signal *number* before it is sent to the program. This is useful when a program being debugged handles signals such as interrupts. Initially all signals are trapped except SIGCONT, SIGCHLD, SIGALRM and SIGKILL.

cont Continue execution from where it stopped. Execution cannot be continued if the process has "finished", that is, called the standard procedure "exit". *Dbx* does not allow the process to exit, thereby letting the user to examine the program state.

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.

Displaying and Naming Data

print *expression* [, *expression* ...]
Print out the values of the expressions. Array expressions are always subscripted by brackets ("[ ]"). Variables having the same identifier as one in the current block may be referenced as "*block-name* . *variable*". The field reference operator (".") can be used with pointers as well as records, making the C operator "->" unnecessary (although it is supported). The construct *expression* \ *typename* can be used to print the *expression* out in the format of the type named *typename*.

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.

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.

assign *variable* = *expression*
set *variable* = *expression*
Assign the value of the expression to the variable.

call *procedure(parameters)*
Execute the object code associated with the named procedure or function. Currently, calls to a procedure with a variable number of arguments are not possible. Also, string parameters are not passed properly for C.

where Print out a list of the active procedures and function.

dump [> *filename*]
Print the names and values of all active variables.

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.

Accessing Source Files

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.

Machine Level Commands

tracei [*address*] [**if** *cond*]
tracei [*variable*] [**at** *address*] [**if** *cond*]
stopi [*address*] [**if** *cond*]
stopi [**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 no address is specified, 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 "$rN" where N is the number of the register. Addresses may be expressions made up of other addresses and the operators "+", "-", and indirection (unary "*").

Miscellaneous Commands

sh *command-line*
Pass the command line to the shell for execution. The SHELL environment variable determines which shell is used.

alias *new-command-name old-command-name*
Respond to *new-command-name* as though it were *old-command-name.*

help Print out a synopsis of *dbx* commands.

gripe Invoke a mail program to send a message to the person in charge of *dbx.*

source *filename*
Read *dbx* commands from the given *filename.*

quit Exit *dbx.*

FILES

a.out	object file
.dbxinit	initial commands

SEE ALSO

cc(1), f77(1), mod(1)

COMMENTS

Non-local gotos can cause some trace/stops to be missed. Most of the command names are too long. The alias facility helps, but is really quite weak. A *csh*-like history capability would improve the situation. But then, who wants to duplicate the c-shell in a debugger?

Dbx suffers from the same "multiple include" malady as does sdb. If you have a program consisting of a number of object files and each is built from source files that include header files, the symbolic information for the header files is replicated in each object file. Since about one debugger start-up is done for each link, having the linker (ld) re-organize the symbol information won't save much time, though it would reduce some of the disk space used. The problem is an artifact of the unrestricted semantics of #include's in C; for example an include file can contain static declarations that are separate entities for each file in which they are included.

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. 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. The following constructions are recognized:

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 and in registers are printed.

q
: exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, 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 executed 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 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* for array operations.

An example which prints the first ten values of n! is

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1), which is a preprocessor for *dc* providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

DIAGNOSTICS

'x is unimplemented' where x is an octal number.
'stack empty' for not enough elements on the stack to do what was asked.
'Out of space' when the free list is exhausted (too many digits).
'Out of headers' for too many numbers being kept around.
'Out of pushdown' for too many items on the stack.
'Nesting Depth' for too many levels of nested execution.

NAME

dcat − convert troff phototypesetter output files to press format and print them on the Dover.

SYNOPSIS

dcat [options] [files]

DESCRIPTION

The program *dcat* reads in phototypesetter files (produced by *troff*(1) with the −t option), converts them to press format, and ships them to the Dover. The press files are usually fairly large, however, and cause the Dover to grunt hard when printing them.

If no files are given, *dcat* reads from the standard input. Thus *dtroff*(1) causes *troff* to create a press file by giving it the −t option and piping the output into *dcat*.

The environment variable DCAT may be used to specify defaults. The value of DCAT is parsed as a string of arguments before the arguments that appear on the command line. For example, DCAT='−n "Reagan, Ronald"' sets your user name to "Reagan, Ronald".

The possible options are:

−b *banner* Use *banner* to label the output. It will appear on the cover page on the line labeled "File:".

−c *n* Cause *n* copies of the output to be produced. The default is −c 1.

−n *name* Cause *name* to be used as the delivery address of your output (the "For:" field on the cover sheet). This usually defaults to your full name, causing output to be filed by the first character of your first name. If you'd rather go by your last name, then use −n "Bovik, Harold". You can arrange to have this done by default by using the DCAT environment variable.

−ox *distance* Move all characters *distance* to the right. The default is −ox 0.

−oy *distance* Move all characters *distance* upward. The default is −oy 0.

−p *file* Write the press file on *file* and do not ship it to the Dover. Press files produced by *dcat* consume a rather large amount of disk space, however, and should not be saved on disk as a long term policy.

−q Suppresses idle chit-chat on stderr; normally, *dcat* gives page number and font information, but *troff* users who depend upon diversions to stderr for creation of index entries will want to avoid contamination.

−s Do not attempt to please Spruce by substituting fonts. Normally, *dcat* will read the Spruce fonts listing file and will substitute a similar font if one of the exact desired size and face cannot be found.

−tx *distance* Set the maximum tolerable horizontal character positioning error.

−ty *distance* Set the maximum tolerable vertical character positioning error.

The desired position for each character on the page is computed by tracking phototypesetter motion commands. This position is compared with the actual position which resulted from the last character written to the press file. For various reasons, not the least of which is that the positioning units of the phototypesetter and of press files are incommensurate, the desired position and the actual position will probably not be identical. −tx allows up to *distance* of absolute error in the horizontal component of a character position before an explicit set-x command is emitted into the press file. −ty does the same for the vertical component.

Note that the actual position can only be estimated, because the width information in the fonts.widths file is only an insufficiently accurate approximation to what Spruce actually does. Allowing more tolerance causes the generated press file to be smaller because fewer set-x and set-y commands need to be emitted. The default is −tx −1 −ty 0, which causes a set-x command to

be emitted for every character.

Distances are specified by giving a fixed-point real number optionally followed by a units measure. For example, 1in means one inch, 15mm means fifteen millimeters, and 0.3feet means three-tenths of a foot. If no units are indicated, micas (which are 2540 to an inch) are assumed.

ENVIRONMENT

DCAT	string of options to be used by *dcat*.
FONTFILE	the path name of a file to use in preference to /usr/stanford/lib/fonts.widths
SPRUCEFONTS	the path name of a file to use in preference to /usr/stanford/lib/spruce.fonts

FILES

/usr/stanford/lib/fonts.widths	width information for all of the available fonts.
/usr/stanford/lib/spruce.fonts	an exact listing of all of the fonts available on Spruce.
/usr/tmp/Dcat XXXXXX	default press file output.

SEE ALSO

cz(1), dtroff(1), dpr(1), dpq(1), troff(1)

DIAGNOSTICS

Lots, but they should be self explanatory.

BUGS

No check is made for overflow when computing the internal representation of a distance measure specified in an option.

Entity list lengths are not checked. It turns out that Spruce cannot print a press file which contains an entity list over 32767 bytes long. It is not legal for an entity list to be over 65535 bytes long.

Signals are not caught. This means that if you interrupt *dcat*, it will quit, leaving a partially-written press file on the disk.

dcat attemps to perform a faithful translation from the Bell Laboratories Times Roman phototypesetter character set into the available fonts on the Dover. This is not always possible, particularly in the larger point sizes.

dcat determines where page breaks should occur by assuming that each page is exactly 11 inches long. There is no indication of page boundaries in the phototypesetter command script. Needless to say, should you change the length of a page, *dcat* will no longer paginate the output properly.

The Spruce fonts listing is not currently available; you should use the −s option to avoid hassles. (Actually, the code for this is disabled, so −s is effectively the default.)

HISTORY

29-Apr-81 Jeffrey Mogul at Stanford University
Added description of −s switch. Modified for Stanford realities.

24-Mar-81 Tom Rodeheffer (tlr) at Carnegie-Mellon University
Updated the description of −s switch, now that the Spruce fonts listing file has been implemented.

06-Mar-81 Tom Rodeheffer (tlr) at Carnegie-Mellon University
Added bug description for the current behavior with the version 3 fonts most of whose entries in the fonts.widths file are scalable.

27-Feb-81 Mike Accetta (mja) at Carnegie-Mellon University
Width tables which resemble the widths of the Dover fonts are now installed for *troff* to use. See the command *dtroff*, which runs *troff* with these width tables and pipes the result through *dcat*.

12-Feb-81 Tom Rodeheffer (tlr) at Carnegie-Mellon University

Fixed the −b, −c, and −n options, which had been being parsed but ignored. Memory allocation failure is now checked for.

29-Jan-81 Tom Rodeheffer (tlr) at Carnegie-Mellon University
The new CAT translation tables are finally installed. We have as many fonts as needed: fonts are loaded only on demand. This version released for public use.

12-Dec-80 Mike Accetta (mja) at Carnegie-Mellon University
dcat created by merging a CAT => DVI converter and a DVI => PRESS converter. The latter was written by W. I. Nowicki of Stanford University.

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. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

option	*values*
if=	input file name; standard input is default
of=	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs;* also, if no conversion is specified, it is particularly efficient since no copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input records before starting copy
files=*n*	copy *n* input files before terminating (makes sense only where input is a magtape or similar device).
seek=*n*	seek *n* records from beginning of output file before copying
count=*n*	copy only *n* input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
block	convert variable length records to fixed length
unblock	convert fixed length records to variable length
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
... , ...	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.

Cbs is used only if *ascii, unblock, ebcdic, ibm,* or *block* conversion is specified. In the first two cases, *cbs* characters are placed into the conversion buffer, any specified character mapping is done, trailing blanks trimmed and new-line added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer, and blanks added to make up an output record of size *cbs.*

After completion, *dd* reports the number of whole and partial input and output blocks.

For example, to read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file *x:*

```
dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magtape. *Dd* is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

SEE ALSO

cp(1), tr(1)

DIAGNOSTICS

f+p records in(out): numbers of full and partial records read(written)

BUGS

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The 'ibm' conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

One must specify "conv=noerror,sync" when copying raw disks with bad sectors to insure *dd* stays synchronized.

NAME

ddt68, fddt68 -- symbolic debugger for 68000

SYNOPSIS

fddt68 b.out
cc68 ... -lddt (Sun)

DESCRIPTION

fddt68 is a symbolic disassembler for b.out files created by the 68000 linker (ld68). Its main purpose is to allow testing of ddt logic in a more hospitable environment than the 68000. It also gives a way of inspecting the assembly language form of a program without having to produce a .s file. In addition it gives a check on the operation of as68 and ld68. It is called by typing:

fddt68 *filename*

on the Vax.

ddt68 is a symbolic debugger for the 68000. It is loaded at link edit time with the cc68 flag -lddt. On starting a program with ddt loaded the user will be at the ddt command level. Breakpoints may be set, and the program started, using the commands described below.

COMMANDS

ddt68 recognizes the following commands ($ is used for <*esc*>):

expression/

expression\

open the location at *expression* and display the contents in the current typeout mode. The user may then optionally type an expression, whose value replaces the contents of the open location. Finally the location is *closed* by typing one of *return* (to return to *ddt*'s main command loop), / (to open the next location), or \ (to open the previous location).

expression$g

go - plant any breakpoints set with the *$b* command, load the registers, and start execution at *expression*. If *expression* is unspecified or zero, execution resumes starting from the current value of *$pc* (normally the point where the program was last interrupted).

expression$x

execute the next *expression* instructions, starting from the current value of *$pc* and printing out all executed instructions. If *expression* is omitted, 1 is assumed.

expression$$x

same as above except execute subroutine calls and traps as single instructions, i.e. do not descend into the called subroutine.

expression$p

proceed - like *go* with no argument, except that if we are presently at a breakpoint then *expression* counts the number of times to pass this breakpoint before breaking. *1$p* is synonymous with *$g*.

expression$*bno*b

set breakpoint *bno* (in the range 1-9) at *expression*. If *bno* is omitted the first unused breakpoint number is assigned (the commonest usage). If *expression* is 0 the named breakpoint is cleared, or if there is no named breakpoint (*bno* is omitted) all breakpoints are cleared. If *expression* is omitted all breakpoints are printed, whether or not *bno* is present.

$*rspec*/

$*rspec*\ examine register *rspec* where *rspec* is one of:

d0-d7 data registers 0-7

a0-a7 address registers 0-7

fp frame pointer (synonym for *a6*)

sp stack pointer (synonym for *a7*)

pc program counter

sr status register

expression$=

type out *expression* in current output radix.

lowlimit<*highlimit*>*pattern*?

search for *pattern* in the range *lowlimit* (inclusive) to *highlimit* (exclusive). The pattern is interpreted as an object of the type in force as the current typeout mode, with instructions and strings being treated as 2-byte words. Objects are assumed to be aligned on word (2-byte) boundaries except for 1-byte types and strings which are aligned on byte boundaries. A mask (set with the following command) determines how much of the pattern is significant in the search, except that if the pattern is a string constant a separate mask matched to the length of the string is used. The three arguments to the search command are sticky; thus if *lowlimit*< (resp. *highlimit*>) is omitted, the most recent lowlimit (resp. highlimit) applies. While *pattern* may be omitted, the final ? may not be omitted.

expression$m

set the search mask to *expression*. *-1$m* forces a complete match, *f$m* checks only the low order 4 bits, *0$m* will make the search pattern match anything.

base$ir set input radix to *base*. (Note *10$i* can never change the radix.) If *base* is omitted hexadecimal is assumed.

base$or

set output radix to *base*. If *base* is omitted hexadecimal is assumed.

$*type*t temporarily set typeout mode to *type* where *type* is one of:

<space>

deduce type from type of nearest symbol

c type out bytes as ascii characters.

h type out bytes in current output radix.

w type out words in current radix.

l type out longs in current radix.

s type out strings in current radix. (In this mode new values cannot be entered.)

i type out as 68000 symbolic instructions. (In this mode only the first two bytes of the opened location may be changed; the new value is typed in as a numeric expression rather than as a symbolic instruction.)

The new typeout mode stays in effect until a *return* is typed.

$$*type*t permanently set typeout mode to *type*.

An *expression* is composed of symbols, numeric constants, string constants, and the operators +, -, and / representing 2's complement addition, subtraction, and inclusive bitwise or. Symbols are delimited by operators or <*esc*>. A string constant has from 1 to 4 characters which are packed right justified into one long to form a numeric constant; thus "did"=646A64. String constants are particularly useful in conjunction witht the search command for searching for a string. The single character . (dot) as a symbol on its own represents the address of the currently open memory location. All operations are carried out using 32 bit arithmetic and evaluated strictly left to right.

AUTHORS

Jim Lawson and Vaughan Pratt

NAME

deroff − remove nroff, troff, tbl and eqn constructs

SYNOPSIS

deroff [**−w**] file ...

DESCRIPTION

Deroff reads each file in sequence and removes all *nroff* and *troff* command lines, backslash constructions, macro definitions, *eqn* constructs (between '.EQ' and '.EN' lines or between delimiters), and table descriptions and writes the remainder on the standard output. *Deroff* follows chains of included files ('.so' and '.nx' commands); if a file has already been included, a '.so' is ignored and a '.nx' terminates execution. If no input file is given, *deroff* reads from the standard input file.

If the **−w** flag is given, the output is a word list, one 'word' (string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed) per line, and all other characters ignored. Otherwise, the output follows the original, with the deletions mentioned above.

SEE ALSO

troff(1), eqn(1), tbl(1)

BUGS

Deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most errors result in too much rather than too little output.

NAME

detex – remove TeX constructs

SYNOPSIS

detex file ...

DESCRIPTION

Detex reads each file in sequence and removes all *TeX* control sequences and math mode constructions, producing a list of the words on standard output, one word per line. If no arguments are given, standard input is read. A 'word' is a sequence of letters, apostrophes, and discretionary hyphens. Apostrophes and discretionary hyphens are removed. Single letter words and words all in uppercase letters are not output, nor are any other characters.

The intended use is as a filter before *spell* (1).

SEE ALSO

spell(1), tex(1), tex78(1)

BUGS

It is assumed that only dollar signs are used to indicate math mode.

HISTORY

Installed at Stanford by Howard Trickey, June 1983.

NAME

df − disk free

SYNOPSIS

df [**−i**] [filesystem ...] [file ...]

DESCRIPTION

Df prints out the amount of free disk space available on the specified *filesystem,* e.g. "/dev/rp0a", or on the filesystem in which the specified *file,* e.g. "$HOME", is contained. If no file system is specified, the free space on all of the normally mounted file systems is printed. The reported numbers are in kilobytes.

Other options are:

−i Report also the number of inodes which are used and free.

FILES

/etc/fstab list of normally mounted filesystems

SEE ALSO

fstab(5), icheck(8), quot(8)

NAME

diction,explain − print wordy sentences; thesaurus for diction

SYNOPSIS

diction [−ml] [−mm] [−n] [−f pfile **]** file ...
explain

DESCRIPTION

Diction finds all sentences in a document that contain phrases from a data base of bad or wordy diction. Each phrase is bracketed with [ ]. Because *diction* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package **−ms** may be overridden with the flag **−mm.** The flag **−ml** which causes **deroff** to skip lists, should be used if the document contains many lists of non-sentences. The user may supply her/his own pattern file to be used in addition to the default file with **−f** *pfile.* If the flag **−n** is also supplied the default file will be suppressed.

Explain is an interactive thesaurus for the phrases found by diction.

SEE ALSO

deroff(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks. In particular, *diction* doesn't grok **−me.**

NAME

diff − differential file and directory comparator

SYNOPSIS

diff [**−l**] [**−r**] [**−s**] [**−cefh**] [**−b**] dir1 dir2
diff [**−cefh**] [**−b**] file1 file2
diff [**−D***string*] [**−b**] file1 file2

DESCRIPTION

If both arguments are directories, *diff* sorts the contents of the directories by name, and then runs the regular file *diff* algorithm (described below) on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed. Options when comparing directories are:

−l long output format; each text file *diff* is piped through *pr*(1) to paginate it, other differences are remembered and summarized after all text file differences are reported.

−r causes application of *diff* recursively to common subdirectories encountered.

−s causes *diff* to report files which are the same, which are otherwise not mentioned.

−Sname
starts a directory *diff* in the middle beginning with file *name*.

When run on regular files, and when comparing text files which differ during directory comparison, *diff* tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, *diff* finds a smallest sufficient set of file differences. If neither *file1* nor *file2* is a directory, then either may be given as '−', in which case the standard input is used. If *file1* is a directory, then a file in that directory whose file-name is the same as the file-name of *file2* is used (and vice versa).

There are several options for output format; the default output format contains lines of these forms:

n1 a *n3,n4*
n1,n2 d *n3*
n1,n2 c *n3,n4*

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging 'a' for 'd' and reading backward one may ascertain equally 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 '>'.

Except for **−b**, which may be given with any of the others, the following options are mutually exclusive:

−e producing a script of *a*, *c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. 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 *diff* need be on hand. A 'latest version' appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Extra commands are added to the output when comparing directories with **−e**, so that the result is a *sh*(1) script for converting text files which are common to the two directories from their state in *dir1* to their state in *dir2*.

−f produces a script similar to that of **−e**, not useful with *ed*, and in the opposite order.

-c produces a diff with lines of context. The default is to present 3 lines of context and may be changed, e.g to 10, by **-c10**. With **-c** the output format is modified slightly: the output beginning with identification of the files involved and their creation dates and then each change is separated by a line with a dozen *'s. The lines removed from *file1* are marked with '-'; those added to *file2* are marked '+'. Lines which are changed from one file to the other are marked in both files with '!'.

-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.

-Dstring
causes *diff* to create a merged version of *file1* and *file2* on the standard output, with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *file1*, while defining *string* will yield *file2*.

-b causes trailing blanks (spaces and tabs) to be ignored, and other strings of blanks to compare equal.

FILES

/tmp/d?????
/usr/lib/diffh for **-h**
/bin/pr

SEE ALSO

cmp(1), cc(1), comm(1), ed(1), diff3(1)

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some, 2 for trouble.

BUGS

Editing scripts produced under the **-e** or **-f** option are naive about creating lines consisting of a single '.'.

When comparing directories with the **-b** option specified, *diff* first compares the files ala *cmp*, and then decides to run the *diff* algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string differences.

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* = *n2*, 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.

Under the **−e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, *i.e.* the changes that normally would be flagged **====** and **====3**. Option **−x** (**−3**) produces a script to incorporate only changes flagged **====** (**====3**). The following command will apply the resulting script to 'file1'.

```
(cat script; echo '1,$p') | ed − file1
```

FILES

/tmp/d3?????
/usr/lib/diff3

SEE ALSO

diff(1)

BUGS

Text lines that consist of a single '.' will defeat **−e**.

NAME

dist – redistribute a message to additional addresses

SYNOPSIS

dist [+folder] [msg] [–form formfile] [–editor editor] [–annotate] [–noannotate] [–inplace] [–noinplace] [–help]

DESCRIPTION

Dist is similar to *forw*. It prepares the specified message for redistribution to addresses that (presumably) are not on the original address list. The file "distcomps" in the user's MH directory, or a standard form, or the file specified by '–form formfile' will be used as the blank components file to be prepended to the message being distributed. The standard form has the components "Distribute-to:" and "Distribute-cc:". When the message is sent, "Distribution-Date: date", "Distribution-From: name", and "Distribution-Id: id" (if '–msgid' is specified to *send*;) will be prepended to the outgoing message. Only those addresses in "Distribute-To", "Distribute-cc", and "Distribute-Bcc" will be sent. Also, a "Distribute-Fcc: folder" will be honored (see *send*;).

Send recognizes a message as a redistribution message by the existence of the field "Distribute-To:", so don't try to redistribute a message with only a "Distribute-cc:".

If the '–annotate' switch is given, each message being distributed will be annotated with the lines:

Distributed: date
Distributed: Distribute-to: names

where each "to" list contains as many lines as required. This annotation will be done only if the message is sent directly from *dist*. If the message is not sent immediately from *dist* (i.e., if it is sent later via *send*;), "comp –use" may be used to re-edit and send the constructed message, but the annotations won't take place. The '–inplace' switch causes annotation to be done in place in order to preserve links to the annotated message.

See *comp* for a description of the '–editor' switch and for options upon exiting from the editor.

If a +folder is specified, it will become the current folder, and the current message will be set to the message being redistributed.

FILES

/etc/mh/components	The message skeleton
or <mh-dir>/components	Rather than the standard skeleton
$HOME/mh_profile	The user profile
<mh-dir>/draft	The default message file
/usr/bin/send	To send the composed message

PROFILE COMPONENTS

Path:	To determine the user's MH directory
Editor:	To override the use of /bin/ned as the default editor
<lasteditor>–next:	editor to be used after exit from <lasteditor>

DEFAULTS

'+folder' defaults to the current folder
'msg' defaults to cur
'–editor' defaults to /bin/ned

'–noannotate'
'–noinplace'

NAME

dl68 – b.out -> .dl downloader component of cc68

SYNOPSIS

dl68 [-T -v -o -s] filename

DESCRIPTION

Dl68 is a downloader for the Motorola 68000 Design Module. It takes its input, a b.out format file, from filename and in the absence of the -o option sends its output to stdout.

If there are any symbols these are loaded, starting at 0x6BA on vm (the Design Module) or 0x1F000 on v1 (the Sun1 prototype). The start and end of the symbol table are stored at 0x570 and 0x574 respectively on either board.

The options are:

-T *textorigin*
specifies where the text (code) is to be loaded.

-v*n* specifies the board version. Default is v1 (Sun1 prototype). vm denotes the Motorola Design Module.

-o *filename*
specifies the output file. Defaults to stdout.

-s*DE* specifies the *data/end* record types to generate. The default is s28, 24 bit addresses. The s19 format, 16 bit addresses, is used by the Data I/O programmers.

FILES

/usr/sun/ld68/down.c /usr/bin/dl68

NAME

dlx – download with error correction – 68000 Sun1 monitor

SYNOPSIS

dlx filename

DESCRIPTION

Dlx downloads to the Sun1 monitor a file produced with the -d option of *cc68(1)*. After sending each record (i.e. one line) dlx checks the monitor's acknowledgment and retransmits the record if necessary.

Dlx is intended to be invoked from the Sun1 monitor, so precede with l, as in

l dlx *filename*

Dlx is "smart" about the *filename* argument, to a certain extent. If the filename includes an extension (i.e., '.' embedded in the tail of the pathname), then it is taken verbatim. Otherwise, if the filename specified does not exist (or exists, but is not in S-record format), *dlx* will attempt to use *filename.dl* instead. In any case, it will give up if the file it finally chooses is not in S-record format.

The *dlx* program first does stty cbreak -echo. It then prints out '\\' followed by some commands that cause echoing of channel B (host) input on channel A (tty) and send monitor output to channel B so that *dlx* can read it. It then waits for two >'s, which is what the monitor will send to the host after the channel switch. Then it sends the records from the input file. After sending each record it checks that the response was Y>.

If the response was anything but Y> it assumes the worst and tries to get everything back into synch. First it sends BEL (control G) to alert the user that there has been an error. Then it sends a unique hex id (starting from 5007e as it happens, and counting up by 1 for each erroneous record), followed by return and q (for quit), then looks for this id to be printed out by the monitor (which it does as part of opening this location). When the unique id finally arrives, it skips to the next >, and resends the record.

On exit from *dlx*, whether caused by interrupt or termination, reset (of the Vax tty) is performed, and control is passed back to the monitor via i A. If the monitor is not working correctly it may be necessary to interrupt *dlx* (with ↑C) to return to the Vax shell.

The echoing of the text on the user's terminal contains no linefeeds, so that each record overwrites the preceding one.

AUTHORS

V. R. Pratt, Jeffrey Mogul

BUGS

If the synchronizing unique id is garbled in either direction *dlx* will hang. This may require resetting the monitor.

NAME

dpq -- prints the Dover printer queue

SYNOPSIS

dpq [-t]

DESCRIPTION

Dpq prints the Dover printer queue. Each entry in the queue is printed showing the owner of the queue entry, an identification number, the size of the entry in characters, the file which is to be printed, and the tag specified when queued. The *id* is useful for removing a specific entry from the printer queue using *dprm*(1).

If the -t option is given, the time that the file was queued is printed instead of the name of the spooler temporary file.

The program also attempts to determine the status of the printing server (default server-name is Tahoe.)

FILES

/usr/spool/dpd/* Spool area

SEE ALSO

dpr(1), dprm(1)

BUGS

Needs to be able to handle multiple servers.

NAME

dpr -- dover printer spooler

SYNOPSIS

dpr [−m] [−n] [−t tagname] [−r] [name ...]

DESCRIPTION

Dpr causes the named files to be queued for printing on the Dover. If no files are named, the standard input is read. The options are:

-m	causes notification via *mail*(1) to be sent when the job completes.
-n	causes a message to be written to your terminal when the job completes.
-r	causes the named file(s) to be removed after being printed.
-t tagname	causes tagname to be the 'tag' associated with the print job in the queue. You can find out what this tag is with *dpq*(1) and the notification sent by −m or −n includes this tag. If you do not specify a tag, it defaults to the last filename given; also, only the last tag specified is used. If no files are named, the tag is blank.

If the file is a press file in PDP-11 byte order, it will be byte-swapped during transmission; otherwise, it will be sent without byteswapping.

FILES

/usr/spool/dpd/*	spool area
/usr/lib/dpd	printer daemon
/usr/lib/dwrite	used to send message to terminal

SEE ALSO

dpq(1), dprm(1)

BUGS

Queued jobs print in directory (seemingly random) order.

This program inherits any undiscovered bugs of lpr.

If it looks like nothing is getting sent to the printer, try removing the file /usr/spool/dpd/lock (if it is there.) Then, run /usr/lib/dpd. Do not do this if a process named dpd is already running.

The −n option is stupid about which terminal to write to if you are logged in more than once. In particular, it behaves the same way as *write*(1) ; that is, it writes to the terminal named in the last entry in /etc/utmp that you are logged in on.

Should handle multiple servers.

NAME

dprm − remove a file from the Dover printer queue

SYNOPSIS

dprm [id ...] [filename ...] [owner ...]

DESCRIPTION

Dprm removes an entry from the Dover printer queue. The id, filename or owner should be that reported by *dpq*. All apropriate files will be removed. The id of each file removed from the queue will be printed.

SEE ALSO

dpr(1), dpq(1)

FILES

/usr/spool/dpd/*

BUGS

NAME

dtree – print directory tree structures

SYNOPSIS

dtree [–adfglnpsvx] [–b filenamesize] [–c linelength] [directory1 ...]

DESCRIPTION

Dtree is a program to print out the tree structure of directories and their children. If no directories are specified, *dtree* takes the current working directory to be the top of the tree structure. It prints out just the directory structure by default. If no flags are specified, *dtree* prints out just the directory structures. Recognized options are as follows:

-a Include files in printout (excluding entries beginning with '.')

-b Take the next argument to be the maximum length of a directory name; default is 14 characters, or the value associated with a -c argument, if any. any directories with names longer than this length will not be searched, thus any files and directories within them will not be included in the output.

-c Take the next argument to be the length of each column of the printout. (By default, this is 14, the maximum filename length. Any lengths greater than the column width are truncated accordingly, and the last character which fits into the column is replaced by an asterisk.)

-d List directories first. For each directory, its subdirectories will be listed first, and then all its other entries.

-f List files first. Reverse of -d.

-l Long listing. Useful information is printed to the right of each entry. The name of the owner, its size in blocks, and its mode are printed.

-g Same as the -l flag, except that the group name is used instead of the owner name. If both the -l and -g flags are used, both the owner and group will be printed.

-n No sort. Names are listed in the order they are read from the directory.

-p Include entries beginning with '.' (excluding "." and "..").

-s Simplify the long listing. Prints uid, size in blocks, and octal mode of the file. This flag implies the -l flag unless the -g flag is specified.

-v Allow for columns to be of variable length. Rather than using the same width for each column of output, each column is shortened as much as possible without truncating any names.

-x Do not cross file systems. Dtree will not cross over to a subdirectory if it is on a different file system.

AUTHOR

Dave Borman, Digital Unix Engineering Group
decvax!borman
Originally written at St. Olaf College, Northfield, MN.

NAME

dtroff – troff to the Dover

SYNOPSIS

dtroff [**−l***length*] troff arguments

DESCRIPTION

Dtroff runs *troff*(1) in an environment to produce typeset output on the Dover. It uses the *deat*(1) program to convert *troff* output to press format, then *dpr* to spool the output to the Dover. The **−l** (lower case l) option causes the output to be split onto successive pages every *length* inches rather than the default 11 inches.

Currently all fonts are mapped into times roman, with some special symbols coming out of the math font.

AUTHOR

Bill Nowicki, Jeff Mogul

FILES

/usr/lib/tmac/tmac.veat	default font mounts and bug fixes
/usr/stanford/lib/font	directory containing fonts

SEE ALSO

troff(1), deat(1), dpr(1)

BUGS

Since some macro packages work correctly only if the fonts named R, I, B, and S are mounted, and since the Dover fonts have different widths for individual characters than the fonts found on the typesetter, there are a number of problems with the font files.

NAME

du − summarize disk usage

SYNOPSIS

du [**−s**] [**−a**] [name ...]

DESCRIPTION

Du gives the number of kilobytes contained in all files and, recursively, directories within each specified directory or file *name*. If *name* is missing, '.' is used.

The argument **−s** causes only the grand total to be given. The argument **−a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

A file which has two links to it is only counted once.

SEE ALSO

df(1), quot(8)

BUGS

Non-directories given as arguments (not under **−a** option) are not listed.
If there are too many distinct linked files, *du* counts the excess files multiply.

NAME

dumpfonts – show what Press fonts are available in fonts.widths

SYNOPSIS

dumpfonts [fontwidthfile]

DESCRIPTION

Dumpfonts is used to show what fonts are available for use with programs that understand Press format (primarily *cz*(1), *dvip*(1), and *dcat*(1)). If an argument is given, it is taken as the filename of the font widths file; otherwise, the standard font widths file is used.

The information given includes the font's "family name", its face ('b' for bold, 'i' for italic), the octal values of the first and last characters in the font, the font size in points (scalable fonts are indicated), and the rotation if present.

FILES

/usr/stanford/lib/fonts.widths standard font width information

SEE ALSO

cz (1)

BUGS

The format of the listing could be compressed and/or sorted better.

NAME

dviboise – send DVI files to the HP2680a printer using TCP

SYNOPSIS

dviboise [options] [files]

DESCRIPTION

The *dviboise* program sends DVI files such as produced by TeX and LaTeX to the Boise printer. It has the same options as boise(1), but only the following are meaningful:

-b banner	Uses "banner" to label the output. It will appear on the cover page on the line labeled "File:".
-c n	Causes n copies of the output to be printed. The default is to print one copy.
-n name	Causes the given name to be used as the delivery address of your output (the "For user:" field on the cover sheet). This defaults to your full name.
-v	Verbose mode. *Dviboise* will print a line on the standard error describing what was done with each file.
-s server	Specifies what server the files are to be sent to. *Server* may be the name of any Internet host. The default is, of course, "boise".
-p port	Specifies which port on the server machine the files are to be sent to, in decimal. The default is 35. It is probably not useful to change this.

SEE ALSO

boise(1), pr(1), cat(1), tex(1), latex(1)

DIAGNOSTICS

Most are self-explanatory. The message "Connection timed out" means the printer server did not respond, probably because it is down.

FEATURES

The server now understands the line-drawing \specials, but not the rectangle one.

NAME

dviimp -- convert DVI files to impress format

SYNOPSIS

dviimp [options] file

DESCRIPTION

dviimp converts *file*, which should be in DVI format, to IMPRESS format for printing on an Imprint-10 printer. The options are:

- **-r** Remove the input file when done.
- **-c***n* Prints *n* copies.
- **-b** The next argument is printed on the banner page to identify this listing.
- **-M***n* Set the Imprint-10 memory parameter to *n*, where $1 \leq n \leq 5$. See the Imprint-10 Programmer's Guide for a description of this value.
- **-i** The next argument is the name of the output file.
- **-v** Produces more verbose output.
- **-d** Produces extensive output for debugging.

If the **-i** flag is not used, the output is written to a temporary file and, upon completion of processing, *ipr* is called to spool the output file for printing. Since the **-b** and **-c** flags are merely passed to *ipr*, they have no effect if **-i** is used.

FILES

/usr/spool/ipd/xf??????
/usr/local/fonts/imagen/raster/*

SEE ALSO

iprint(1), catdvi(1), ipr(1)

DIAGNOSTICS

'No font set' means no font command appears before the first printed output on a page. 'Page contains too much data' means the page will not fit in the Imprint-10's internal memory. 'Character *c* not in font *f*' and 'Character *n* out of range' mean the character is not defined in the font. 'Bad DVI version *n*' means the input file is not a version 1 DVI file. 'Font *f* version *n*' means this font file is not a version 0 RAS-format file. Other diagnostics should be self-explanatory.

Voluminous output will be generated by using the **-v** and **-d** flags.

AUTHOR

Imagen Corp.

NAME

dvip, dvid – convert a dvi (TeX output) file to press format.

SYNOPSIS

dvip dvifile[.dvi] [-p pressfile] [-d]

dvid dvifile

DESCRIPTION

The program *dvip* reads in a "DVI" file (produced by *TEX(1)*) and converts it to press format. If no extension is given for dvifile, a '.dvi' is added. The name of the press file created is formed by replacing the '.dvi' extension with '.press'. Or, the press file name can be given explicitly with the *-p pressfile* option.

The *-d* option instructs *dvip* to enter a dialog with the user to gather additional options. Currently, these are the starting page number, maximum number of pages, magnification, and margins. The starting page number is given in terms of the values of the TeX counters 0 to 9 associated with each page. The specification c0.c1. ... means the first page where \count0=c0, \count1=c1, etc. A '*' means "any", so for example 2.*.-3 is the first page where \count0=2 and \count2=-3.

The command *dvid filename* can be used to handle the most common use of dvip. It causes *filename.press* to be created from *filename.dvi* and then immediately shipped to the Dover printer (with *cz* (1)). Afterwards, *filename.press* is removed.

Dvip understands some extended commands that allow TeX to draw textured rectangles and polygonal lines. For that purpose, *dvip* keeps a table of 256 points numbered 0 to 255. All pointer coordinates are initialized to 0. that is, points are set to the top left corner of the page. Point numbers can be reused at any time. Point coordinates are remembered through page changes, thus making it easy to refer to exactly the same position in several different pages.

To generate these commands in TeX one must use \special strings. Numeric arguments are in decimal and the case of command words is important. The TeX commands that invoke them are:

\special{point <number>}
: Remember the current position under the given index.

\special{join <pensize> <number1> <number2> .. <number i>}
: Draw straight line segments joining points <number1> and <number2>, <number2> and <number3>, etc., using a pen of thickness <pensize> (in units of dover pixels). It only recognizes certain sizes (2, 4, 6, 8, 12, 16, 32). The smallest size is not known to work properly. Any size not in the list is converted to 4.

\special{rectangle <number1> <character> <number2> <number3>}
: The four low order bits of <number1> describe the lower row of a 4*4 square bit pattern. The next higher four bits describe the third row from the top of that bit pattern, and so on for the next two groups of four bits. The bitmap given in this way will be replicated to fill the rectangular area whose opposite corners are point <number2> and point <number3>. The character argument must be 'o' at the moment, and it means to do the logical "or" of the new rectangle with the previous page contents (same as overstriking). CAVEAT: the Dover Spruce server can only handle bitmaps of certains sizes.

FILES

/usr/stanford/lib/tex82/fonts metric information for all of the available fonts.

SEE ALSO

cz(1), dpr(1), dpq(1)

DIAGNOSTICS

Lots, but they should be self explanatory.

HISTORY

September 1983, UNIX Pascal version by Howard Trickey, after the SAIL version by Ignazio Zabala.

NAME

echo – echo arguments

SYNOPSIS

echo [**–n**] [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a newline on the standard output. If the flag **–n** is used, no newline is added to the output.

Echo is useful for producing diagnostics in shell programs and for writing constant data on pipes. To send diagnostics to the standard error file, do 'echo ... 1>&2'.

NAME

ed − text editor

SYNOPSIS

ed [−] [−x] [name]

DESCRIPTION

Ed is the standard text editor.

If a *name* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed's* buffer so that it can be edited. If **−x** is present, an *x* command is simulated first to handle an encrypted file. The optional − suppresses the printing of explanatory output and should be used when the standard input is an editor script.

Ed operates on a copy of any file it is editing; changes made in 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*.

Commands to *ed* have a simple and regular structure: zero or more *addresses* followed by a single character *command*, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Missing addresses are supplied by default.

In general, only one command may appear on a line. Certain commands allow the addition of text to 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.

Ed supports a limited form of *regular expression* notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. In the following specification for regular expressions the word 'character' means any character but newline.

1. Any character except a special character matches itself. Special characters are the regular expression delimiter plus \ [. and sometimes ^ * $.
2. A . matches any character.
3. A \ followed by any character except a digit or ( ) matches that character.
4. A nonempty string *s* bracketed [*s*] (or [^ *s*]) matches any character in (or not in) *s*. In *s*, \ has no special meaning, and] may only appear as the first letter. A substring *a*−*b*, with *a* and *b* in ascending ASCII order, stands for the inclusive range of ASCII characters.
5. A regular expression of form 1-4 followed by * matches a sequence of 0 or more matches of the regular expression.
6. A regular expression, *x*, of form 1-8, bracketed \(*x* \) matches what *x* matches.
7. A \ followed by a digit *n* matches a copy of the string that the bracketed regular expression beginning with the *n*th \(matched.
8. A regular expression of form 1-8, *x*, followed by a regular expression of form 1-7, *y* matches a match for *x* followed by a match for *y*, with the *x* match being as long as possible while still permitting a *y* match.
9. A regular expression of form 1-8 preceded by ^ (or followed by $), is constrained to matches that begin at the left (or end at the right) end of a line.
10. A regular expression of form 1-9 picks out the longest among the leftmost matches in a line.
11. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in one command (see *s* below) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\'. This also applies to the character bounding the regular expression (often '/') and to '\' itself.

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; however, the exact effect on the current line is discussed under the description of the command. Addresses are constructed as follows.

1. The character '.' addresses the current line.
2. The character '$' addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. '*x*' addresses the line marked with the name *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A regular expression enclosed in slashes '/' addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the beginning of the buffer.
6. A regular expression enclosed in queries '?' addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the end of the buffer.
7. An address followed by a plus sign '+' or a minus sign '−' followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.
8. 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'.
9. If an address ends with '+' or '−', then 1 is added (resp. subtracted). As a consequence of this rule and rule 8, the address '−' refers to the line before the current line. Moreover, trailing '+' and '−' characters have cumulative effect, so '−−' refers to the current line less 2.
10. To maintain compatibility with earlier versions of the editor, the character '^' in addresses is equivalent to '−'.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma ','. They may also be separated by a semicolon ';'. In this case the current line '.' is set to the previous address before the next address is interpreted. 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 following the line corresponding to the first address. The special form '%' is an abbreviation for the address pair '1,$'.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the default.

As mentioned, it is generally illegal for more than one command to appear on a line. However, most commands may be suffixed by 'p' or by 'l', in which case the current line is either printed or listed respectively in the way discussed below. Commands may also be suffixed by 'n',

meaning the output of the command is to be line numbered. These suffixes may be combined in any order.

(.)a
<text>
.

The append command reads the given text and appends it after the addressed line. '.' is left on the last line input, if there were any, otherwise at the addressed line. Address '0' is legal for this command; text is placed at the beginning of the buffer.

(.,.)c
<text>
.

The change command deletes the addressed lines, then accepts input text which replaces these lines. '.' is left at the last line input; if there were none, it is left at the line preceding the deleted lines.

(.,.)d

The delete command deletes the addressed lines from the buffer. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e filename

The edit 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. The number of characters read is typed. 'filename' is remembered for possible use as a default file name in a subsequent *r* or *w* command. If 'filename' is missing, the remembered name is used.

E filename

This command is the same as *e*, except that no diagnostic results when no *w* has been given since the last buffer alteration.

f filename

The filename command prints the currently remembered file name. If 'filename' is given, the currently remembered file name is changed to 'filename'.

(1,$)g/regular expression/command list

In the global command, the first step is to mark every line which matches the given regular expression. Then for every such line, the given command list is executed with '.' initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must be ended with '\'. *A*, *i*, and *c* commands and associated input are permitted; the '.' terminating input mode may be omitted if it would be on the last line of the command list. The commands *g* and *v* are not permitted in the command list.

(.)i
<text>
.

This command inserts the given text before the addressed line. '.' is left at the last line input, or, if there were none, at the line before the addressed line. This command differs from the *a* command only in the placement of the text.

(.,.+1)j

This command joins the addressed lines into a single line; intermediate newlines simply disappear. '.' is left at the resulting line.

(.)k*x*

The mark command marks the addressed line with name *x*, which must be a lower-case

letter. The address form "*x*' then addresses this line.

(.,.)l
The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in two-digit octal, and long lines are folded. The *l* command may be placed on the same line after any non-i/o command.

(.,.)m*a*
The move command repositions the addressed lines after the line addressed by *a*. The last of the moved lines becomes the current line.

(.,.)n
The number command prints the addressed lines with line numbers and a tab at the left.

(.,.)p
The print command prints the addressed lines. '.' is left at the last line printed. The *p* command may be placed on the same line after any non-i/o command.

(.,.)P
This command is a synonym for *p*.

q The quit command causes *ed* to exit. No automatic write of a file is done.

Q This command is the same as *q*, except that no diagnostic results when no *w* has been given since the last buffer alteration.

($) r filename
The read command reads in the given file after the addressed line. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). The file name is remembered if there was no remembered file name already. 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 left at the last line read in from the file.

(.,.) s/regular expression/replacement/ or,
(.,.) s/regular expression/replacement/g
The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all matched strings are replaced by the replacement specified, 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. It is an error for the substitution to fail on all addressed lines. Any punctuation character may be used instead of '/' to delimit the regular expression and the replacement. '.' is left at the last line substituted.

An ampersand '&' appearing in the replacement is replaced by the string matching the regular expression. The special meaning of '&' in this context may be suppressed by preceding it by '\'. The characters '*n*' where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression enclosed between '\(' and '\)'. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left.

Lines may be split by substituting new-line characters into them. The new-line in the replacement string must be escaped by preceding it by '\'.

One or two trailing delimiters may be omitted, implying the 'p' suffix. The special form 's' followed by *no* delimiters repeats the most recent substitute command on the addressed lines. The 's' may be followed by the letters **r** (use the most recent regular expression for the left hand side, instead of the most recent left hand side of a substitute command), **p** (complement the setting of the *p* suffix from the previous substitution), or **g** (complement the setting of the *g* suffix). These letters may be combined in any order.

(.,.)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 on the last line of the copy.

(.,.) u
The undo command restores the buffer to it's state before the most recent buffer modifying command. The current line is also restored. Buffer modifying commands are *a, c, d, g, i, k,* and *v*. For purposes of undo, *g* and *v* are considered to be a single buffer modifying command. Undo is its own inverse.

When *ed* runs out of memory (at about 8000 lines on any 16 bit mini-computer such as the PDP-11) This full undo is not possible, and *u* can only undo the effect of the most recent substitute on the current line. This restricted undo also applies to editor scripts when *ed* is invoked with the - option.

(1, $) v/regular expression/command list
This command is the same as the global command *g* except that the command list is executed *g* with '.' initially set to every line *except* those matching the regular expression.

(1, $) w filename
The write command writes the addressed lines onto the given file. If the file does not exist, it is created. The file name is remembered if there was no remembered file name already. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). '.' is unchanged. If the command is successful, the number of characters written is printed.

(1, $) W filename
This command is the same as *w*, except that the addressed lines are appended to the file.

(1, $) wq filename
This command is the same as *w* except that afterwards a *q* command is done, exiting the editor after the file is written.

x A key string is demanded from the standard input. Later *r, e* and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption. (.+1) z or,

(.+1) z*n*
This command scrolls through the buffer starting at the addressed line. 22 (or *n*, if given) lines are printed. The last line printed becomes the current line. The value *n* is sticky, in that it becomes the default for future *z* commands.

($) =
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 *sh*(1) to be interpreted as a command. '.' is unchanged.

(.+1,.+1) <newline>
An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to '.+1p'; it is useful for stepping through text. If two addresses are present with no intervening semicolon, *ed* prints the range of lines. If they are separated by a semicolon, the second line is printed.

If an interrupt signal (ASCII DEL) is sent, *ed* prints '?interrupted' and returns to its command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and, on mini computers, 128K characters in the temporary file. The limit on the number of lines depends on the amount of core: each line takes 2 words.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last newline. It refuses to read files containing non-ASCII characters.

FILES

/tmp/e*
edhup: work is saved here if terminal hangs up

SEE ALSO

B. W. Kernighan, *A Tutorial Introduction to the ED Text Editor*
B. W. Kernighan, *Advanced editing on UNIX*
ex(1), sed(1), crypt(1)

DIAGNOSTICS

'?name' for inaccessible file; '?self-explanatory message' for other errors.

To protect against throwing away valuable work, a *q* or *e* command is considered to be in error, unless a *w* has occurred since the last buffer change. A second *q* or *e* will be obeyed regardless.

BUGS

The *l* command mishandles DEL.
The *undo* command causes marks to be lost on affected lines.
The *x* command, -x option, and special treatment of hangups only work on UNIX.

NAME

efl − Extended Fortran Language

SYNOPSIS

efl [option ...] [filename ...]

DESCRIPTION

Efl compiles a program written in the EFL language into clean Fortran. *Efl* provides the same control flow constructs as does *ratfor*(1), which are essentially identical to those in C:

statement grouping with braces;

decision-making with if, if-else, and switch-case; while, for, Fortran do, repeat, and repeat...until loops; multi-level break and next. In addition, EFL has C-like data structures, and more uniform and convenient input/output syntax, generic functions. EFL also provides some syntactic sugar to make programs easier to read and write:

free form input:

multiple statements/line; automatic continuation statement label names (not just numbers),

comments:

this is a comment

translation of relationals:

>, >=, etc., become .GT., .GE., etc.

return (expression)

returns expression to caller from function

define: define name replacement

include:

include filename

The Efl command option **−w** suppresses warning messages. The option **−C** causes comments to be copied through to the Fortran output (default); **−#** prevents comments from being copied through. If a command argument contains an embedded equal sign, that argument is treated as if it had appeared in an **option** statement at the beginning of the program. *Efl* is best used with *f77*(1).

SEE ALSO

f77(1), ratfor(1).

S. I. Feldman, *The Programming Language EFL*, Bell Labs Computing Science Technical Report #78.

NAME

eftprec – receive-only PUP/EFTP file transfer program with routing

SYNOPSIS

eftprec **hostname filename [du]**

DESCRIPTION

Eftprec is a replacement for *eftp*(1) which provides reception of files via this protocol across a Pup internet. The old *eftp* program is somewhat more flexible (it understands more about text files) but is not capable of routing across gateways, and the table of host names is wired in. The *eftprec* program, on the other hand, does proper Pup routing and hostname lookup, but does not send files, nor does it do end-of-line convention formatting; it merely moves bytes. For sending files via eftp, see *eftpsend*(1).

The *hostname* and *filename* arguments are fairly obvious; the file named by the latter is sent to the host named by the former. (If the *hostname* argument is 0, then eftprec will accept the first file sent by any host.) The optional "key" argument can be any combination of the following letters:

d (debug) – prints out a lot of useless information.

u (unswap) – causes odd and even bytes to be swapped (wrt to network standard byte order, see *byteorder*(9)) during transmission; this is useful when you've got a file to receive to a host which uses the non-standard byte ordering (e.g., another Vax).

Two examples are given to illustrate the use of this program, as a replacement for *eftp*:

eftprec sail file.press
    is roughly the same as
eftp xesq sail file.press

eftprec sail file.press u
    is roughly the same as
eftp xeq sail file.press

AUTHORS

Jeffrey Mogul and Erik Hedberg

SEE ALSO

eftp(1), eftpsend(1)

DIAGNOSTICS

Many self-explanatory diagnostics; typically, they might involve failure to open a network channel, failure to find a route to the host or the address of the host, failure to open the file, or protocol failures.

BUGS

Relatively untested.

Relatively useless.

NAME

eftpsend — send-only PUP/EFTP file transfer program with routing

SYNOPSIS

eftpsend **hostname filename [** du **]**

DESCRIPTION

Eftpsend is a replacement for *eftp*(1) which supports our most important use of this protocol (sending files to a Press-printer) across a Pup internet. The old *eftp* program is somewhat more flexible (it understands more about text files) but is not capable of routing across gateways, and the table of host names is wired in. The *eftpsend* program, on the other hand, does proper Pup routing and hostname lookup, but does not receive files, nor does it do end-of-line convention formatting; it merely moves bytes. For receiving files via eftp, see *eftprec*(1).

The *hostname* and *filename* arguments are fairly obvious; the file named by the latter is sent to the host named by the former. If the filename is "-" then the standard input is used. If the hostname includes a port specification, then the file is sent to that port, instead of the standard one.

The optional "key" argument can be any combination of the following letters:

d (debug) -- prints out a lot of useless information.

u (unswap) — causes odd and even bytes to be swapped (wrt to network standard byte order, see *byteorder*(9)) during transmission; this is useful when you've got a file to send to a host which uses the non-standard byte ordering (e.g., another Vax).

Two examples are given to illustrate the use of this program, as a replacement for *eftp*:

eftpsend dover file.press
    is roughly the same as
eftp resq dover file.press

eftpsend dover file.press u
    is roughly the same as
eftp req dover file.press

AUTHOR

Jeffrey Mogul

SEE ALSO

eftp(1), eftprec(1)

DIAGNOSTICS

Many self-explanatory diagnostics; typically, they might involve failure to open a network channel, failure to find a route to the host or the address of the host, failure to open the file, or protocol failures.

BUGS

Relatively untested.

NAME

emacs — a screen editor

SYNOPSIS

emacs files...

teachemacs

DESCRIPTION

Unix *Emacs* is a text editor styled after the editors of the same name that exist on ITS, Twenex and Multics. Invoking *emacs* with a list of files causes *emacs* to start up and do a visit-file on each of the named files. For more information, read the manual.

Teachemacs is a command which gives the user a hands-on tutorial introduction to *Emacs*. It is nowhere near complete, but should make the manual less mysterious to the naive user, and may be sufficient education for the casual user.

A one-page reference card, suitable for printing on a Press-format printer, is /usr/stanford/doc/EmacsRefCard.press, and can be printed with *cz*(1).

FILES

/usr/stanford/lib/emacs/*

ENVIRONMENT

LOADPATH is a path used when locating files with the load command or using the start-up .com feature. This is /usr/stanford/lib/emacs/maclib here.

SEE ALSO

The Unix Emacs manual, which should be on /usr/stanford/doc/emacs.press on your favorite VAX.

If you don't have a Press-file printer, /usr/stanford/doc/emacs.doc is an ASCII version of the manual.

Changes to Emacs are documented (in reverse chronological order) in /usr/stanford/lib/emacs/ChangeLog.

BUGS

Lines of length 79 are printed on two lines, with \ in column 79.

With track-eol-on-↑N-↑P set to 1, typing ↑E↑P does indeed take you to the end of the preceding line unless you happen to be on the last and empty line of the file. (Picky, picky.)

Emacs's extension language, MLisp, has the following bugs.

(if x y) fails to evaluate x when x is a variable, whence x looks like a string to 'if', resulting in y always being selected.

Comparisons (including equality) between a string and a number, and numeric operations on strings (including the logical operations) yield what would be obtained if the string were replaced by 0. (In all other contexts, the string "23" is treated as identical to the number 23, so this is a point of inconsistency as much as a genuine bug.)

If error-occured [sic] is used to trap errors, the error message is neither printed nor made available, making diagnosis impossible.

HISTORY

08-Jan-81 James Gosling (jag) at Carnegie-Mellon University
Created.

19-Jan-82 Jeffrey Mogul at Stanford University
Added tutorial.

NAME

eqn, neqn, checkeq − typeset mathematics

SYNOPSIS

eqn [**−dxy**] [**−pn**] [**−sn**] [**−fn**] [file] ...
checkeq [file] ...

DESCRIPTION

Eqn is a troff(1) preprocessor for typesetting mathematics on a Graphic Systems phototypesetter, *neqn* on terminals. Usage is almost always

eqn file ... | troff
neqn file ... | nroff

If no files are specified, these programs reads from the standard input. A line beginning with '.EQ' marks the start of an equation; the end of an equation is marked by a line beginning with '.EN'. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; subsequent text between delimiters is also treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument **−d***xy* or (more commonly) with 'delim *xy*' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by 'delim off'. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within *eqn* are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character like *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde ~ represents a full space in the output, circumflex ^ half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup.** Thus *x sub i* makes x_i, *a sub i sup 2* produces a_i^2, and *e sup {x sup 2 + y sup 2}* gives $e^{x^2+y^2}$.

Fractions are made with **over**: *a over b* yields $\frac{a}{b}$.

sqrt makes square roots: *1 over sqrt {ax sup 2 +bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim_{n\to\infty} \sum_0^n x_i$ is made with *lim from {n−> inf} sum from 0 to n x sub i.*

Left and right brackets, braces, etc., of the right height are made with **left** and **right:** *left [x sup 2 + y sup 2 over alpha right]~=~1* produces $\left[x^2+\frac{y^2}{\alpha}\right] = 1$. The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x} = \overline{f(t)}$, *y dotdot bar ~ =~ n under* is $\ddot{\bar{y}} \sim = \sim \underline{n}$, and *x vec ~ =~ y dyad* is $\vec{x} \sim = \sim \breve{y}$.

Sizes and font can be changed with **size** *n* or **size** $\pm n$, **roman**, **italic**, **bold**, and **font** *n*. Size and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **−s***n* and **−f***n*.

Normally subscripts and superscripts are reduced by 3 point sizes from the previous size; this may be changed by the command-line argument **−p***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**: *define thing % replacement %* defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The *%* may be any character that does not occur in *replacement*.

Keywords like *sum* ($\sum$) *int* ($\int$) *inf* (∞) and shorthands like >= ($\geq$) −> ($\rightarrow$), and != ($\neq$) are recognized. Greek letters are spelled out in the desired case, as in *alpha* or *GAMMA*. Mathematical words like sin, cos, log are made Roman automatically. *Troff*(1) four-character escapes like \(bs (⑧) can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff* when all else fails.

SEE ALSO

troff(1), tbl(1), ms(7), eqnchar(7)
B. W. Kernighan and L. L. Cherry, *Typesetting Mathematics—User's Guide*
J. F. Ossanna, *NROFF/TROFF User's Manual*

BUGS

To embolden digits, parens, etc., it is necessary to quote them, as in 'bold "12.3"'.

NAME

error − analyze and disperse compiler error messages

SYNOPSIS

error [−n] [−s] [−q] [−v] [−t suffixlist] [−I ignorefile] [name]

DESCRIPTION

Error analyzes and optionally disperses the diagnostic error messages produced by a number of compilers and language processors to the source file and line where the errors occurred. It can replace the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously without machinations of multiple windows in a screen editor.

Error looks at the error messages, either from the specified file *name* or from the standard input, and attempts to determine which language processor produced each error message, determines the source file and line number to which the error message refers, determines if the error message is to be ignored or not, and inserts the (possibly slightly modified) error message into the source file as a comment on the line preceding to which the line the error message refers. Error messages which can't be categorized by language processor or content are not inserted into any file, but are sent to the standard output. *Error* touches source files only after all input has been read. By specifying the **−q** query option, the user is asked to confirm any potentially dangerous (such as touching a file) or verbose action. Otherwise *error* proceeds on its merry business. If the **−t** touch option and associated suffix list is given, *error* will restrict itself to touch only those files with suffices in the suffix list. Error also can be asked (by specifying **−v**) to invoke *vi*(1) on the files in which error messages were inserted; this obviates the need to remember the names of the files with errors.

Error is intended to be run with its standard input connected via a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into *error*. For example, when using the *csh* syntax,

```
make −s lint |& error −q −v
```

will analyze all the error messages produced by whatever programs *make* runs when making lint.

Error knows about the error messages produced by: *make, cc, cpp, ccom, as, ld, lint, pi, pc* and *f77*. *Error* knows a standard format for error messages produced by the language processors, so is sensitive to changes in these formats. For all languages except *Pascal,* error messages are restricted to be on one line. Some error messages refer to more than one line in more than one files; *error* will duplicate the error message and insert it at all of the places referenced.

Error will do one of six things with error messages.

synchronize
: Some language processors produce short errors describing which file it is processing. *Error* uses these to determine the file name for languages that don't include the file name in each error message. These synchronization messages are consumed entirely by *error*.

discard
: Error messages from *lint* that refer to one of the two *lint* libraries, */usr/lib/llib-lc* and */usr/lib/llib-port* are discarded, to prevent accidently touching these libraries. Again, these error messages are consumed entirely by *error*.

nullify
: Error messages from *lint* can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The names of functions to ignore are taken from either the file named *.errorrc* in the

users's home directory, or from the file named by the **-I** option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.

not file specific
Error messages that can't be intuited are grouped together, and written to the standard output before any files are touched. They will not be inserted into any source file.

file specific Error message that refer to a specific file, but to no specific line, are written to the standard output when that file is touched.

true errors Error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are candidates for inserting into the file they refer to. Other error messages are consumed entirely by *error* or are written to the standard output. *Error* inserts the error messages into the source file on the line preceding the line the language processor found in error. Each error message is turned into a one line comment for the language, and is internally flagged with the string "###" at the beginning of the error, and "%%%" at the end of the error. This makes pattern searching for errors easier with an editor, and allows the messages to be easily removed. In addition, each error message contains the source line number for the line the message refers to. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C or Pascal, it is possible to insert a comment into another comment, which can wreak havoc with a future compilation. To avoid this, programs with comments and source on the same line should be formatted so that language statements appear before comments.

Options available with *error* are:

-n Do *not* touch any files; all error messages are sent to the standard output.

-q The user is *queried* whether s/he wants to touch the file. A "y" or "n" to the question is necessary to continue. Absence of the **-q** option implies that all referenced files (except those referring to discarded error messages) are to be touched.

-v After all files have been touched, overlay the visual editor *vi* with it set up to edit all files touched, and positioned in the first touched file at the first error. If *vi* can't be found, try *ex* or *ed* from standard places.

-t Take the following argument as a suffix list. Files whose suffixes do not appear in the suffix list are not touched. The suffix list is dot separated, and "*" wildcards work. Thus the suffix list:

".c.y.foo*.h"

allows *error* to touch files ending with ".c", ".y", ".foo*" and ".y".

-s Print out *statistics* regarding the error categorization. Not too useful.

Error catches interrupt and terminate signals, and if in the insertion phase, will orderly terminate what it is doing.

AUTHOR

Robert Henry

FILES

~/.errorrc	function names to ignore for *lint* error messages
/dev/tty	user's teletype

BUGS

Opens the teletype directly to do user querying.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's format of error messages may cause *error* to not understand the error message.

Error, since it is purely mechanical, will not filter out subsequent errors caused by 'floodgating' initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected (error puts them before). The alignment of the '|' marking the point of error is also disturbed by *error*.

Error was designed for work on CRT's at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.

NAME

etherport -- show status of ethernet minor devices

SYNOPSIS

etherport

DESCRIPTION

Etherport is used to find out the status of the Ethernet minor devices on the system.

When you run it, you get a list that looks like this:

```
/dev/enet (3mb):        0-15 busy; 16-31 free
/dev/eneta (10mb):      none busy; 0-31 free
64 ethernet minor devices -- 16 busy, 48 free
```

Minor devices listed as "busy" are in use; those listed as "free" are not.

SEE ALSO

enet(4), enstat(8)

AUTHOR

Jeffrey Mogul (after John Seamons)

BUGS

Absolutely none.

NAME

ex, edit − text editor

SYNOPSIS

ex [−] [−v] [−t tag] [−r] [+*command*] [−l] name ...
edit [ex options]

DESCRIPTION

Ex is the root of a family of editors: *edit, ex* and *vi. Ex* is a superset of *ed,* with the most notable extension being a display editing facility. Display based editing is the focus of *vi.*

If you have not used *ed,* or are a casual user, you will find that the editor *edit* is convenient for you. It avoids some of the complexities of *ex* used mostly by systems programmers and persons very familiar with *ed.*

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 which focuses on the display editing portion of *ex.*

DOCUMENTATION

The document *Edit: A tutorial* provides a comprehensive introduction to *edit* assuming no previous knowledge of computers or the UNIX system.

The *Ex Reference Manual − Version 3.5* is a comprehensive and complete manual for the command mode features of *ex,* but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of *ex* see the editing documents written by Brian Kernighan for the editor *ed;* the material in the introductory and advanced documents works also with *ex.*

An Introduction to Display Editing with Vi introduces the display editor *vi* and provides reference material on *vi.* All of these documents can be found in volume 2c of the Programmer's Manual. In addition, the *Vi Quick Reference* card summarizes the commands of *vi* in a useful, functional way, and is useful with the *Introduction.*

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/etc/termcap	describes capabilities of terminals
~/.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ed(1), grep(1), sed(1), grep(1), vi(1), termcap(5), environ(7)

AUTHOR

Originally written by William Joy
Mark Horton has maintained the editor since version 2.7, adding macros, support for many unusual terminals, and other features such as word abbreviation mode.

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 don't 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.

NAME

exlog – extract data from system load log file

SYNOPSIS

exlog [–fhlu] [fromdate **[** todate **]]**

DESCRIPTION

Exlog extracts data from the system load log file logged between *fromdate* and *todate* and outputs it to standard output. *Fromdate* and *todate* are of the form: month/day[/year], where month, day, and year are decimal integers. If not specified, year is assumed to be the current year. If *todate* is not specified, data logged since *fromdate* is extracted. If *fromdate* is specified as '.', data logged on the current date is extracted. If *fromdate* is not specified, all data is extracted.

The default format for the data output is the same binary format input. Other output formats may be specifed with the following options:

–h output a histogram of the load. This is intended to be viewed on the terminal or printed by piping into *cz*(1) with the options '-f gacha6 -2'.

–l output a table of load values.

–u output a table giving numbers of users logged in.

One other option, **–f**, directs the program to act as a filter, reading from standard input instead of the system load log file.

FILES

/usr/adm/loadlog

SEE ALSO

loadlog(1), loadavg(1)

AUTHOR

Marc R. Hannah

BUGS

NAME

expand, unexpand — expand tabs to spaces, and vice versa

SYNOPSIS

expand [−tabstop] [−tab1,tab2,...,tabn] [file ...]
unexpand [**−a**] [file ...]

DESCRIPTION

Expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. *Expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single *tabstop* argument is given then tabs are set *tabstop* spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

Unexpand puts tabs back into the data from the standard input or the named files and writes the result on the standard output. By default only leading blanks and tabs are reconverted to maximal strings of tabs. If the **−a** option is given, then tabs are inserted whenever they would compress the resultant file by replacing two or more characters.

NAME

explain, diction– print wordy sentences; thesaurus for diction

SYNOPSIS

diction [**–ml**] [**–mm**] [**–n**] [**–f** pfile] file ...
explain

DESCRIPTION

Diction finds all sentences in a document that contain phrases from a data base of bad or wordy diction. Each phrase is bracketed with [ ]. Because *diction* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package **–ms** may be overridden with the flag **–mm.** The flag **–ml** which causes **deroff** to skip lists, should be used if the document contains many lists of non-sentences. The user may supply her/his own pattern file to be used in addition to the default file with **–f** *pfile.* If the flag **–n** is also supplied the default file will be suppressed.

Explain is an interactive thesaurus for the phrases found by diction.

SEE ALSO

deroff(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks. In particular, *diction* doesn't grok **–me.**

NAME

expr − evaluate arguments as an expression

SYNOPSIS

expr arg ...

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument.

The operators and keywords are listed below. The list is in order of increasing precedence, with equal precedence operators grouped.

expr | *expr*
: yields the first *expr* if it is neither null nor '0', otherwise yields the second *expr*.

expr & *expr*
: yields the first *expr* if neither *expr* is null or '0', otherwise yields '0'.

expr relop expr
: where *relop is one of* < <= = != >= >, yields '1' if the indicated comparison is true, '0' if false. The comparison is numeric if both *expr* are integers, otherwise lexicographic.

expr + *expr*
expr − *expr*
: addition or subtraction of the arguments.

expr * *expr*
expr / *expr*
expr % *expr*
: multiplication, division, or remainder of the arguments.

expr : *expr*
: The matching operator compares the string first argument with the regular expression second argument; regular expression syntax is the same as that of *ed*(1). The \(...\) pattern symbols can be used to select a portion of the first argument. Otherwise, the matching operator yields the number of characters matched ('0' on failure).

(*expr*)
: parentheses for grouping.

Examples:

To add 1 to the Shell variable *a*:

```
a=`expr $a + 1`
```

To find the filename part (least significant part) of the pathname stored in variable *a*, which may or may not contain '/':

```
expr $a : '.*/\(.*\)' '|' $a
```

Note the quoted Shell metacharacters.

SEE ALSO

sh(1), test(1)

DIAGNOSTICS

Expr returns the following exit codes:

0	if the expression is neither null nor '0',
1	if the expression is null or '0',
2	for invalid expressions.

NAME

eyacc − modified yacc allowing much improved error recovery

SYNOPSIS

eyacc [**−v**] [grammar]

DESCRIPTION

Eyacc is an old version of *yacc*(1), which produces tables used by the Pascal system and its error recovery routines. *Eyacc* fully enumerates test actions in its parser when an error token is in the look-ahead set. This prevents the parser from making undesirable reductions when an error occurs before the error is detected. The table format is different in *eyacc* than it was in the old *yacc*, as minor changes had been made for efficiency reasons.

SEE ALSO

yacc(1)

"Practical LR Error Recovery" by Susan L. Graham, Charles B. Haley and W. N. Joy; SIGPLAN Conference on Compiler Construction, August 1979.

AUTHOR

S. C. Johnson

Eyacc modifications by Charles Haley and William Joy.

BUGS

Pc and its error recovery routines should be made into a library of routines for the new *yacc*.

NAME

f77 − Fortran 77 compiler

SYNOPSIS

f77 [option] ... file ...

DESCRIPTION

F77 is the UNIX Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with '.f' are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with '.o' substituted for '.f'.

Arguments whose names end with '.F' are also taken to be Fortran 77 source programs; these are first processed by the C preprocessor before being compiled by *f77*.

Arguments whose names end with '.r' or '.e' are taken to be Ratfor or EFL source programs respectively; these are first transformed by the appropriate preprocessor, then compiled by f77.

Arguments whose names end with '.c' or '.s' are taken to be C or assembly source programs and are compiled or assembled, producing a '.o' file.

The following options have the same meaning as in *cc*(1). See *ld*(1) for load-time options.

−c Suppress loading and produce '.o' files for each source file.

−g Have the compiler produce additional symbol table information for *dbx*(1). Also pass the **−lg** flag to *ld*(1).

−o output
Name the final output file *output* instead of 'a.out'.

−p Prepare object files for profiling, see *prof*(1).

−pg Causes the compiler to produce counting code in the manner of **−p**, but invokes a run-time recording mechanism that keeps more extensive statistics and produces a *gmon.out* file at normal termination. An execution profile can then be generated by use of *gprof*(1).

−w Suppress all warning messages. If the option is '−w66', only Fortran 66 compatibility warnings are suppressed.

−D*name*=*def*

−D*name*
Define the *name* to the C preprocessor, as if by '#define'. If no definition is given, the name is defined as "1". ('.F' suffix files only).

−I*dir* '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **−I** options, then in directories on a standard list. ('.F' suffix files only).

−O Invoke an object-code optimizer.

−S Compile the named programs, and leave the assembler-language output on corresponding files suffixed '.s'. (No '.o' is created.).

The following options are peculiar to *f77*.

−i2 On machines which support short integers, make the default integer constants and variables short. (**−i4** is the standard value of this option). All logical quantities will be short.

−m Apply the M4 preprocessor to each '.r' file before transforming it with the Ratfor or EFL preprocessor.

−onetrip
Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

−u Make the default type of a variable 'undefined' rather than using the default Fortran rules.

−v Print the version number of the compiler, and the name of each pass as it executes.

−C Compile code to check that subscripts are within declared array bounds.

−F Apply the C, EFL, or Ratfor preprocessors to relevant files, put the result in the file with the suffix changed to '.f', but do not compile.

−E*x* Use the string *x* as an EFL option in processing '.e' files.

−R*x* Use the string *x* as a Ratfor option in processing '.r' files.

−N[qxscn]*nnn*
Make static tables in the compiler bigger. The compiler will complain if it overflows its tables and suggest you apply one or more of these flags. These flags have the following meanings:

q Maximum number of equivalenced variables. Default is 150.

x Maximum number of external names (common block names, subroutine and function names). Default is 200.

s Maximum number of statement numbers. Default is 401.

c Maximum depth of nesting for control statements (e.g. DO loops). Default is 20.

n Maximum number of identifiers. Default is 1009.

−U Do not convert upper case letters to lower case. The default is to convert Fortran programs to lower case except within character string constants.

Other arguments are taken to be either loader option arguments, or F77-compatible object programs, typically produced by an earlier run, or perhaps libraries of F77-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name 'a.out'.

FILES

file.[fFresc]	input file
file.o	object file
a.out	loaded output
/usr/lib/f77pass1	compiler
/lib/f1	pass 2
/lib/c2	optional optimizer
/lib/cpp	C preprocessor
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/usr/lib/libU77.a	UNIX interface library
/usr/lib/libF77_p.a	profiling intrinsic function library
/usr/lib/libI77_p.a	profiling Fortran I/O library
/usr/lib/libU77_p.a	profiling UNIX interface library
/lib/libc.a	C library, see section 3
mon.out	file produced for analysis by prof(1).
gmon.out	file produced for analysis by gprof(1).

SEE ALSO

S. I. Feldman, P. J. Weinberger, *A Portable Fortran 77 Compiler*
D. L. Wasley, *Introduction to the f77 I/O Library*
prof(1), gprof(1), cc(1), ld(1), efl(1), ratfor(1)

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

BUGS

This compiler is still somewhat experimental. The optimizer occasionally makes mistakes; it should be avoided when debugging if apparently incorrect results are obtained. Because of an assembler error, complaints about long branches may occur with very large source files; such errors can be avoided by splitting the sources into smaller sections. If necessary, the old version of *f77* can be resurrected from /usr/src/old.

NAME

false, true — provide truth values

SYNOPSIS

true

false

DESCRIPTION

True and *false* are usually used in a Bourne shell script. They test for the appropriate status "true" or "false" before running (or failing to run) a list of commands.

EXAMPLE

```
while false
do
    command list
done
```

SEE ALSO

csh(1), sh(1), true(1)

DIAGNOSTICS

False has exit status nonzero.

NAME

fed − font editor

SYNOPSIS

fed [−i] [−q] name

DESCRIPTION

Fed is an editor for font files. It is display oriented and must be used on an HP 2648 graphics terminal. Fed does the necessary handshaking to work at 9600 baud on the 2648.

The **−i** flag requests *inverse video mode*, where all dots are dark and the background is bright. This provides a setting similar to the hardcopy output of the plotter, and is useful for fonts such as the shadow font where shading is important.

The **−q** flag requests *quiet mode*, where all graphic output is suppressed. This mode is useful on terminals other than the HP 2648 (assuming you are editing blindly) and for operations such as the # and A commands, since these operations do not make essential use of graphics, and since suppression of the graphic output speeds of *fed* considerably.

FONTS

A font is a collection of up to 256 *glyphs*, each of which is some pattern or design. Glyphs are represented on Unix as a rectangular array of dots, each of which is either dark or blank. Each location in the array is called a *pixel*. There are 200 pixels per inch due to the hardware of the Versatec and Varian plotters.

Each glyph has, in addition to its bit pattern, a *base* and a *width*. The base is a point, typically near the lower left of the array, that represents the logical lower left point of the glyph. The base is not restricted to be within the array, in fact, it is usually a few locations to the left of the edge. The vertical position of the base defines the *baseline*, which is held constant for all glyphs when a line is typeset. Letters with descenders, such as "g", go below the baseline. Other glyphs typically rest on the baseline.

The width is used by *troff(1)* to determine where to place the next glyph. It need not be the same as the width of the array, although it is usually about the same.

The size of the array, location of the base, and the width can vary among glyphs in a font. Fonts where all glyphs have the same width are called *fixed width fonts*, others are *variable width fonts*.

Attributes which do not vary among glyphs include the *font name*, which can be up to 11 alphabetic characters, and the *point size*, which is a positive integer indicating the overall size of the font. A point is 1/72 inch. The point size of a font is the distance, in points, from the top of the tallest glyph to the bottom of the lowest. The software of troff currently restricts point sizes to 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, and 36 point. Normal text is usually 10 point.

Font files conventionally have names of the form

name.pointsize

for example, "bocklin.14" to indicate 14 point bocklin. Fed will look for such a file in both the current directory and /usr/lib/vfont. Vtroff will only look in /usr/lib/vfont.

There is a correspondence between *glyphs* and *characters* in a font. For a given font, each glyph has an ASCII character associated with it. The glyph is obtained in troff by typing the associated character, and in fed glyphs are also referred to by their character. However, it is not required for all characters to have a glyph, fonts never have more than 128 glyphs and usually have fewer.

There is usually a natural correspondence between glyphs and characters. For example, the glyph which is a roman lower case 'a' will generally have the ascii character 'a' as its corresponding character. In the special font, the Greek lower case alpha has 'a' as it's

corresponding character, upper case delta has 'D' as it's corresponding character, etc. However, special fonts such as the chess font have glyphs that do not appear to be related to their corresponding characters.

It is easy to confuse glyphs and characters. Note, however, that the three glyphs roman a, bold **a**, and italic *a*, are all different, yet all three correspond to the character 'a'. When this is multiplied by the large number of font styles and point sizes, there are many glyphs that match a single character, (but only one in a particular font).

FED ORGANIZATION

Fed organizes the screen into 21 *windows* in a 3 by 7 array. Each window is 100 by 100 pixels, meaning that the maximum height and width of a glyph is 100 pixels. Since the HP 2648 has a resolution of 100 dots per inch, glyphs displayed on the screen and printer will be double the actual height and width, even when fully zoomed out. There is a *current window*, which will be marked with a square border. There are two *pens*, called *fine* and *bold*. The fine pen is one pixel wide, the bold pen can range from two pixels to ten pixels in diameter. The default width of the bold pen is taken from the point size implied by the file name. The point size is not otherwise used. There are also fine and bold *erasers*.

There are two locations in the window, called the *cursor* and the *mark*. These tools are used to draw on glyphs.

Sometimes the cursor is on, in which case it is indicated by the hardware graphics cursor of the terminal, a cross. The cursor is considered to be located at the center of the cross. Sometimes the *rubber band line* is turned on, showing the path a line drawn would traverse. This line runs from the mark to the cursor, and is the only way the mark is graphically visible.

COMMANDS

Commands to fed are single characters, sometimes followed by any needed arguments. The commands used by fed were chosen to be as similar to *vi*(1) commands as was reasonable. Another distinction is that certain commands are in upper case. These commands were deliberately made hard to type because they cause a large change in the state of the editor and should not be done by accident. In a few cases there are both upper and lower case commands with the same letter.

Alphanumeric Keypad: Note that this is the keypad on the far right. The graphics keypad on the near right will not work. These keys are each synonyms for other commands. They are arranged in a manner that causes the five arrow keys to behave sensibly, but the others need to be memorized or stickers placed on the keys. They are provided for convenience only, and the user can avoid memorization simply by using the mnemonic letter keys instead.

The layout is as follows:

undo (u)	rezoom ( )	fillin (f)
move (m)	up (k)	draw (d)
left (h)	base (b)	right (l)
setdot (.)	down (j)	cleardot (>)

The arrow keys move the cursor one pixel in the indicated direction. The cursor is turned on if it was off. Note that the alphanumeric keys (far right) must be used. The graphics keys (near right) will appear to move the cursor but it will not be moved internally. The cursor cannot be moved outside the current window.

^*L*: Redraw the screen. This is useful if an I/O error or background process has caused the screen to get messed up.

b: Move the cursor to the base of the window. This is the default location of the cursor.

c: If the cursor is on, turn it off. Otherwise, turn it on.

d: Draw a line from the mark to the cursor. The currently selected tool (fine pen, bold pen, fine eraser, bold eraser) is used. The cursor is turned off. The mark is moved to the location of the cursor.

f: Fill in the current hole. The cursor must be in a completely enclosed empty (white) area. The area is set to black. If this command is invoked on the outside or there are any leaks to the outside, the entire outside will be filled in. (Undo is useful in this case.) Filling in cannot jump diagonals, but can rather only spread in the four orthogonal directions.

g <x>: Get a glyph. X can be any character. The glyph corresponding to x is put in a window, and this window is made the current window. The glyph is centered horizontally in the window. The baseline is located at row 70 from the top of the window. The pen and cursor are placed at the base, and the cursor is turned off. The glyph must exist.

h, *j*, *k*, and *l* are accepted to mean left, down, up, and right, respectively. They are synonymous with the alphanumeric arrow keys. They have the same meanings as in *vi(1)*.

m: Move the mark to the current location of the cursor. The cursor is turned on.

n <x>: New glyph. This is similar to *g*, except that the glyph must *not* exist. It is used to create a new glyph. A blank window is created, centered at (50, 70) as in g.

p: Print the contents of the screen. An HP 2631 printer must be connected to the terminal. The screen is copied to the printer. If in inverse video mode, the screen is changed to normal video mode before the print, and then changed back after the print.

r: If the rubber band line is on, turn it off. Otherwise, turn it on.

s <what> [<where>]: Set <what> to <where>. What and where are single characters. The possibilities are:

spf: Set pen fine. ('l' for light is also accepted.)

spb: set pen bold. ('h' for heavy is also accepted.)

sd: Set draw. The pen is used instead of the eraser.

se: Set erase. The eraser is used instead of the pen.

ss<n>: Set size of bold pen. <n> is a digit from 1 to 9. The size of the bold pen is set accordingly. This also affects the bold eraser.

u: Undo. The previous change to the current window is undone. Note that undo is on a window by window basis, so that commands that affect characters or more than one window cannot be undone.

z <n>: Zoom to level n. The screen is blown up by a factor of n. This only affects the appearance of the screen to make it easy to see the individual dots, and does not affect the size of the glyph or the result of a print command. Zooming to 1 shows the entire screen, a level of 3 or 4 is probably good for editing glyphs. When a message is printed on the screen, fed automatically zooms out to level 1 so you can read the message. Hitting space will zoom back. z followed by <return> zooms out without changing the previous zoom.

space: Zoom back to the level most recently requested by the z command.

A <i/e/r> <first> <last> [<oldps> <newps>]:
Artificially italicize/embolden/resize a range of glyphs in the current font. Enter i for italicize, e for embolden, or r for resize, and the first and last character in the range desired. If you are resizing you will also have to enter the old and new point size, each terminated by a return. Each glyph is gotten and changed on the screen visibly. Glyphs are italicized by slanting them to the right at a slope of 1/5. They are emboldened by smearing them to the right a number if pixels equal to the current heavy pen size. They are resized with an algorithm which translates

all on bits to the new position. These operations will be considerably faster if the **-q** option is in effect, since much overhead is involved in the graphic display.

B: Move the base to the cursor. The cursor is turned on.

C <from> <to>: Copy the glyph in character <from> to character <to>. If <from> has a window on the screen, that window is given to <to>.

D <from> <through>: Delete a range of characters in the font, from <from> through <through> inclusive. To delete a single character type it twice.

E <file>: Edit the named file. If changes have been made to the current file, confirmation will be requested. (Either 'y' or 'E' is accepted.) The file name is terminated with return.

F <first> <last>: Show the font on the screen. The characters in the specified range are shown. The width values are used to get natural spacing. The display will remain until another command is typed, at which time the previous display will be redrawn and the new command will be executed. As a special case, a "p" command will print the results of the "F" command instead of the previous display.

I <h/v>: Invert the current glyph about a horizontal or vertical axis, as indicated by *h* or *v*. The axis runs up the center of the window. The base can be subsequently positioned with the *B* command.

K: Kill the current glyph. All dots are set to blank. The glyph is not removed from the font. This is used for redrawing a glyph from scratch or replacing it with another glyph.

M <from> <to>: Move a glyph from <from> to <to>. This is just like the copy command but the original is deleted.

N <file>: Write out the current file, if necessary, and edit the new file specified. The file name is terminated with return.

P <first> <last> <file>: Partial read from a file. A file and the first and last characters in the range are prompted for. Characters not in the range are left unmodified, characters in the range are handled as in the R command.

Q: Quit the editor, without saving any work. If changes have been made confirmation will be required (either 'Q' or 'y' is taken as 'yes'.)

R <file>: Read in the named file on top of the current file. Glyphs are merged wherever possible. If there is a conflict, you will be asked whether fed should take the glyph from the file (f) or buffer (b). Responding with F or B will lock in that mode for the remainder of the read. The file name is terminated with a return.

T <text>:

Typeset the line of text on the terminal. This is similar to the F command except that the given text is arranged on the screen, so you can see how some particular combination of characters would look.

V: Toggle whether editing is being done in inverse video mode.

W <file>: Write the buffer out onto the named file, which is terminated by return. A null file name means the current file name.

ZZ: Exit fed. A write is done, if necessary, followed by a quit. This is the normal way to leave fed. The Z must be doubled for compatibility with *vi*.

.: Turn on the dot under the cursor. The cursor is turned off.

>: Turn off the dot under the cursor. The cursor is turned off.

<char> *<field>* *<value>*: Edit a numerical field. This only makes sense if the glyph has not been gotten (*g* or *n*) yet, since otherwise the values are taken from window specific things such as the base. Fed does not do any sanity checking, but just substitutes the value input. Fields are the first letter of any field from the dispatch structure (see vfont(5)), specifically, these fields are *addr*, *nbytes*, *left*, *right*, *up*, *down*, and *width*. The number, which may be signed, is terminated by a newline.

FILES

/usr/lib/vfont/*.*

SEE ALSO

vfont(5), vfontinfo(1), vtroff(1), vwidth(1)

AUTHOR

Mark Horton

BUGS

Attempting to use the second 128 characters would be folly. Fed has never been tested on such fonts, and at a bare minimum there would be problems trying to input 8 bit characters.

The character DEL is interpreted by the tty driver to mean interrupt. Hence the corresponding glyph cannot be accessed. The *start*, *stop*, and *quit* characters are turned off, but other characters used by the new tty driver must be quoted with ^V.

Changed widths are not copied to the width table used by troff. This only matters if logical widths are changed, or if glyphs are moved around. For these cases, *vwidth(1)* must be used.

The artificial operations don't do a very good job. The quality possible from blowing a font up is in general poor. Italicizing tends to make edges that were previously slanted very ragged. However, these operations are better than nothing at all and are a reasonable first approximation for hand fixing.

The HP 2648 Terminal on which this runs has been stolen.

NAME

file – determine file type

SYNOPSIS

file file ...

DESCRIPTION

File 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.

BUGS

It often makes mistakes. In particular it often suggests that command files are C programs.

Does not recognize Pascal or LISP.

NAME

find − find files

SYNOPSIS

find pathname-list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each pathname in the *pathname-list* (i.e., one or more pathnames) seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, 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 filename
: True if the *filename* argument matches the current file name. Normal Shell argument syntax may be used if escaped (watch out for '[', '?' and '*').

−perm onum
: True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared: *(flags&onum)==onum*.

−type c
: True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **f** or **l** for block special file, character special file, directory, plain file, or symbolic link.

−links n
: True if the file has *n* links.

−user uname
: True if the file belongs to the user *uname* (login name or numeric user ID).

−group gname
: True if the file belongs to group *gname* (group name or numeric group ID).

−size n
: True if the file is *n* blocks long (512 bytes per block).

−inum n
: True if the file has inode number *n*.

−atime n
: True if the file has been accessed in *n* days.

−mtime n
: True if the file has been modified in *n* days.

−exec command
: True if the executed command returns a zero value as exit status. The end of the command must be punctuated by an escaped semicolon. A command argument '{}' is replaced by the current pathname.

−ok command
: Like **−exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response **y**.

−print
: Always true; causes the current pathname to be printed.

−newer file
: True if the current file has been modified more recently than the argument *file*.

The primaries may be combined using the following operators (in order of decreasing precedence):

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).
2) The negation of a primary ('!' is the unary *not* operator).
3) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

4) 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

SEE ALSO

sh(1), test(1), fs(5)

BUGS

The syntax is painful.

NAME

fing – front end for finger

SYNOPSIS

fing [*options*] [*fingeropts*] namestr ...

DESCRIPTION

Fing is a front end for *finger*(1) that understands about system-wide mail aliases and networks. Its functionality is an exact superset of *finger*; any flags or arguments for *finger* will work with *fing*.

In addition, the *namestr* argument may be a system-wide mailing list (e.g. *staff*), or may be of the form *user@host* or *@host*, where *host* is an Internet hostname. For a mailing list, each of the persons on the list will be *finger*ed. For *user@host*, a network connection will be attempted to *host*, and *user* will be *finger*ed there. An argument of the form *@host* will cause a connection to the host, and get a listing of all users logged on there.

If a user has requested that all his mail be forwarded to a particular host by having an alias of the form *user@host*, in the system aliases file, *fing* requests for that user will be forwarded to that host.

Options for *fing* beyond those of *finger* include:

–h Normally, *fing* does a *finger* at each hop along the resolution path. With the **–h** flag, *finger* is run only at the ultimate destination.

–n Don't resolve system-wide aliases.

–v Verbose output; for example, print out the network address of hosts being connected to.

–x Don't expand system-wide aliases after resolving them.

SEE ALSO

finger(1)

FILES

/usr/lib/aliases.* system alias database

AUTHOR

Christopher A. Kent

NOTE

Because of restrictions on the information that can be handed to network finger servers, the VAX server will always invoke finger with the **–m** flag, so that cross-network fingers on someone's name, rather than their login id, may not work, unless there is an appropriate alias.

NAME

finger − user information lookup program

SYNOPSIS

finger [options] name ...

DESCRIPTION

By default *finger* lists the login name, full name, terminal name and write status (as a '*' before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each current UNIX 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 a list of peoples names is given. (Account names as well as first and last names of users are accepted.) This format is multi-line, and includes all the information described above as well as the user's home directory and login shell, any plan which the person has placed in the file *.plan* in their home directory, and the project on which they are working from the file *.project* also in the home directory.

Finger options include:

−m Match arguments only on user name.

−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 login times
~/.plan	plans
~/.project	projects

SEE ALSO

w(1), who(1)

AUTHOR

Earl T. Cohen

BUGS

Only the first line of the *.project* file is printed.

The encoding of the gcos field is UCB dependent − it knows that an office '197MC' is '197M Cory Hall', and that '529BE' is '529B Evans Hall'.

A user information data base 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.

NAME

fmt – simple text formatter

SYNOPSIS

fmt [name ...]

DESCRIPTION

Fmt is a simple text formatter which reads the concatenation of input files (or standard input if none are given) and produces on standard output a version of its input with lines as close to 72 characters long as possible. The spacing at the beginning of the input lines is preserved in the output, as are blank lines and interword spacing.

Fmt is meant to format mail messages prior to sending, but may also be useful for other simple tasks. For instance, within visual mode of the *ex* editor (e.g. *vi*) the command

```
!}fmt
```

will reformat a paragraph, evening the lines.

SEE ALSO

nroff(1), mail(1)

AUTHOR

Kurt Shoens

BUGS

The program was designed to be simple and fast – for more complex operations, the standard text processors are likely to be more appropriate.

NAME

fold – fold long lines for finite width output device

SYNOPSIS

fold [–width] [file ...]

DESCRIPTION

Fold is a filter which will fold the contents of the specified files, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using *expand*(1) before coming to *fold*.

SEE ALSO

expand(1)

BUGS

If underlining is present it may be messed up by folding.

NAME

folder – set/list current folder/message

SYNOPSIS

folder [+folder] [msg] [–all] [–fast] [–nofast] [–up] [–down] [–header] [–noheader] [–total] [–nototal] [–pack] [–nopack] [–help]

folders <equivalent to 'folder –all'>

DESCRIPTION

Since the MH environment is the shell, it is easy to lose track of the current folder from day to day. ***Folder*** will list the current folder, the number of messages in it, the range of the messages (low-high), and the current message within the folder, and will flag a selection list or extra files if they exist. An example of the output is:

```
inbox+ has 16 messages ( 3– 22); cur= 5.
```

If a '+folder' and/or 'msg' are specified, they will become the current folder and/or message. An '–all' switch will produce a line for each folder in the user's MH directory, sorted alphabetically. These folders are preceded by the read-only folders, which occur as mh_profile "cur–" entries. For example,

```
            Folder      # of messages range ); cur msg (other files)
  /fsd/rs/m/tacc  has  35 messages  1– 35); cur= 23.
/rnd/phyl/Mail/EP has  82 messages  1–108); cur= 82.
              ff has   4 messages  1–  4); cur=  1.
          inbox+ has  16 messages  3– 22); cur=  5.
              mh has  76 messages  1– 76); cur= 70.
           notes has   2 messages  1–  2); cur=  1.
            ucom has 124 messages  1–124); cur=  6; (select).

              TOTAL= 339 messages in  7 Folders.
```

The "+" after inbox indicates that it is the current folder. The "(select)" indicates that the folder ucom has a selection list produced by ***pick***. If "others" had appeared in parentheses at the right of a line, it would indicate that there are files in the folder directory that don't belong under the MH file naming scheme.

The header is output if either an '–all' or a '–header' switch is specified; it is suppressed by '–noheader'. Also, if ***folder*** is invoked by a name ending with "s" (e.g., ***folders***), '–all' is assumed. A '–total' switch will produce only the summary line.

If '–fast' is given, only the folder name (or names in the case of '–all') will be listed. (This is faster because the folders need not be read.)

The switches '–up' and '–down' change the folder to be the one above or below the current folder. That is, "folder –down" will set the folder to "<current–folder>/select", and if the current folder is a selection-list folder, "folder –up" will set the current folder to the parent of the selection-list. (See ***pick*** for details on selection-lists.)

The '–pack' switch will compress the message names in a folder, removing holes in message numbering.

FILES

$HOME/mh_profile	The user profile
/bin/ls	To fast-list the folders

PROFILE COMPONENTS

Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder

DEFAULTS

'+folder' defaults to the current folder
'msg' defaults to none
'–nofast'
'–noheader'
'–nototal'
'–nopack'

CONTEXT

If '+folder' and/or 'msg' are given, they will become the current folder and/or message.

NAME

forw – forward messages

SYNOPSIS

forw [+folder] [msgs] [–editor editor] [–form formfile] [–annotate] [–noannotate] [–inplace] [–noinplace] [–help]

DESCRIPTION

Forw may be used to prepare a message containing other messages. It constructs the new message from the components file or '–form formfile' (see *comp*(1)), with a body composed of the message(s) to be forwarded. An editor is invoked as in *comp*, and after editing is complete, the user is prompted before the message is sent.

If the '–annotate' switch is given, each message being forwarded will be annotated with the lines

Forwarded: date
Forwarded: To: names
Forwarded: cc: names

where each "To:" and "cc:" list contains as many lines as required. This annotation will be done only if the message is sent directly from *forw*. If the message is not sent immediately from *forw*, "comp –use" may be used in a later session to re-edit and send the constructed message, but the annotations won't take place. The '–inplace' switch permits annotating a message in place in order to preserve its links.

See *comp* for a description of the '–editor' switch.

FILES

/etc/mh/components	The message skeleton
or <mh-dir>/components	Rather than the standard skeleton
$HOME/mh_profile	The user profile
<mh-dir>/draft	The default message file
/usr/bin/send	To send the composed message

PROFILE COMPONENTS

Path:	To determine the user's MH directory
Editor:	To override the use of /bin/ned as the default editor
Current-Folder:	To find the default current folder
<lasteditor>–next:	editor to be used after exit from <lasteditor>

DEFAULTS

'+folder' defaults to the current folder
'msgs' defaults to cur
'–editor' defaults to /bin/ned
'–noannotate'
'–noinplace'

CONTEXT

If a +folder is specified, it will become the current folder, and the current message will be set to the first message being forwarded.

NAME

fp — Functional Programming language compiler/interpreter

SYNOPSIS

fp

DESCRIPTION

Fp is an interpreter/compiler that implements the applicative language proposed by John Backus. It is written in **FRANZ LISP**.

In a functional programming language intent is expressed in a mathematical style devoid of assignment statements and variables. Functions compute by value only; there are no side-effects since the result of a computation depends solely on the inputs.

Fp "programs" consist of *functional expressions* — primitive and user-defined *fp* functions combined by *functional forms*. These forms take functional arguments and return functional results. For example, the composition operator '@' takes two functional arguments and returns a function which represents their composition.

There exists a single operation in *fp* — *application*. This operation causes the system to evaluate the indicated function using the single argument as input (all functions are monadic).

GETTING STARTED

Fp invokes the system. *Fp* compiles functions into *lisp*(1) source code; *lisp*(1) interprets this code (the user may compile this code using the liszt (1) compiler to gain a factor of 10 in performance). *Control D* exits back to the shell. *Break* terminates any computation in progress and resets any open file units. *)help* provides a short summary of all user commands.

FILES

/usr/ucb/lisp	the FRANZ LISP interpreter
/usr/ucb/liszt	the liszt compiler
/usr/doc/fp	the User's Guide

SEE ALSO

lisp(1), liszt(1).

The Berkeley FP user's manual, available on-line. The language is described in the August 1978 issue of *CACM* (Turing award lecture by John Backus).

BUGS

If a non-terminating function is applied as the result of loading a file, then control is returned to the user immediately, everything after that position in the file is ignored.

FP incorrectly marks the location of a syntax error on large, multi-line function definitions or applications.

AUTHOR

Scott B. Baden

NAME

fpr − print Fortran file

SYNOPSIS

fpr

DESCRIPTION

Fpr is a filter that transforms files formatted according to Fortran's carriage control conventions into files formatted according to UNIX line printer conventions.

Fpr copies its input onto its output, replacing the carriage control characters with characters that will produce the intended effects when printed using *lpr*(1). The first character of each line determines the vertical spacing as follows:

Character	Vertical Space Before Printing
Blank	One line
0	Two lines
1	To first line of next page
+	No advance

A blank line is treated as if its first character is a blank. A blank that appears as a carriage control character is deleted. A zero is changed to a newline. A one is changed to a form feed. The effects of a "+" are simulated using backspaces.

EXAMPLES

a.out | fpr | lpr

fpr < f77.output | lpr

AUTHOR

Robert P. Corbett

BUGS

Results are undefined for input lines longer than 170 characters.

NAME

from − who is my mail from?

SYNOPSIS

from [**−s** sender] [user]

DESCRIPTION

From prints out the mail header lines in your mailbox file to show you who your mail is from. If *user* is specified, then *user*'s mailbox is examined instead of your own. If the -s option is given, then only headers for mail sent by *sender* are printed.

FILES

/usr/spool/mail/*

SEE ALSO

biff(1), mail(1), prmail(1)

NAME

fsplit − split a multi-routine Fortran file into individual files

SYNOPSIS

fsplit [**-e** efile] ... [file]

DESCRIPTION

Fsplit takes as input either a file or standard input containing Fortran source code. It attempts to split the input into separate routine files of the form *name.f*, where *name* is the name of the program unit (e.g. function, subroutine, block data or program). The name for unnamed block data subprograms has the form *blkdtaNNN.f* where NNN is three digits and a file of this name does not already exist. For unnamed main programs the name has the form *mainNNN.f*. If there is an error in classifying a program unit, or if *name.f* already exists, the program unit will be put in a file of the form *zzzNNN.f* where *zzzNNN.f* does not already exist.

Normally each subprogram unit is split into a separate file. When the *-e* option is used, only the specified subprogram units are split into separate files. E.g.:

```
fsplit -e readit -e doit prog.f
```

will split readit and doit into separate files.

DIAGNOSTICS

If names specified via the *-e* option are not found, a diagnostic is written to *standard error.*

AUTHOR

Asa Romberger and Jerry Berkman

BUGS

Fsplit assumes the subprogram name is on the first noncomment line of the subprogram unit. Nonstandard source formats may confuse *fsplit.*

It is hard to use *-e* for unnamed main programs and block data subprograms since you must predict the created file name.

NAME

ftp – file transfer program

SYNOPSIS

ftp [**-v**] [**-d**] [**-i**] [**-n**] [**-g**] [**host**]

DESCRIPTION

Ftp is the user interface to the ARPANET standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* will immediately attempt to establish a connection to an FTP server on that host; otherwise, *ftp* will enter its command interpreter and await instructions from the user. When *ftp* is awaiting commands from the user the prompt "ftp>" is provided the user. The following commands are recognized by *ftp*:

! Invoke a shell on the local machine.

append *local-file* [*remote-file*]
Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for *type*, *format*, *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*.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

close Terminate the FTP session with the remote server, and return to the command interpreter.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified it is used 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, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, output comes to the terminal.

form *format*
Set the file transfer *form* to *format*. The default format is "file".

get *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine. The current settings for *type*, *form*, *mode*, and *structure* are used while transferring the file.

hash Toggle hash-sign ("#") printing for each data block transferred. The size of a data block is 1024 bytes.

glob Toggle file name globbing. With file name globbing enabled, each local file or pathname is processed for *csh*(1) metacharacters. These characters include "*?[]~{}".

Remote files specified in mutliple item commands, e.g. *mput*, are globbed by the remote server. With globbing disabled all files and pathnames are treated literally.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, the output is sent to the terminal.

mdelete *remote-files*
Delete the specified files on the remote machine. If globbing is enabled, the specification of remote files will first be expanded using *ls*.

mdir *remote-files local-file*
Obtain a directory listing of multiple files on the remote machine and place the result in *local-file*.

mget *remote-files*
Retrieve the specified files from the remote machine and place them in the current local directory. If globbing is enabled, the specification of remote files will first be expanding using *ls*.

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Obtain an abbreviated listing of multiple files on the remote machine and place the result in *local-file*.

mode [*mode-name*]
Set the file transfer *mode* to *mode-name*. The default mode is "stream" mode.

mput *local-files*
Transfer multiple local files from the current local directory to the current working directory on the remote machine.

open *host* [*port*]
Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, *ftp* will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see below).

prompt Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default), any *mget* or *mput* will transfer all files.

put *local-file* [*remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for bye.

quote *arg1 arg2 ...*
The arguments specified are sent, verbatim, to the remote FTP server. A single FTP reply code is expected in return.

recv *remote-file* [*local-file*]
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*.

rmdir *directory-name*
Delete a directory on the remote machine.

send *local-file* [*remote-file*]
A synonym for put.

sendport
Toggle the use of PORT commands. By default, *ftp* will attempt to use a PORT command when establishing a connection for each data transfer. If the PORT command fails, *ftp* will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they've been accepted.

status Show the current status of *ftp*.

struct [*struct-name*]
Set the file transfer *structure* to *struct-name*. By default "stream" structure is used.

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*. If no type is specified, the current type is printed. The default type is 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 an account field is not specified, and the FTP server requires it, the user will be prompted for it. Unless *ftp* is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

verbose
Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]
A synonym for help.

Command arguments which have embedded spaces may be quoted with quote (") marks.

FILE NAMING CONVENTIONS

Files specified as arguments to *ftp* commands are processed according to the following rules.

1) If the file name "−" is specified, the **stdin** (for reading) or **stdout** (for writing) is used.

2) If the first character of the file name is "|", the remainder of the argument is interpreted as a shell command. *Ftp* then forks a shell, using *popen*(3) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. ""| ls -lt"". A particularly useful example of this mechanism is: "dir |more".

3) Failing the above checks, if "globbing" is enabled, local file names are expanded according to the rules used in the *csh*(1); c.f. the *glob* command.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The *type* may be one of "ascii", "image" (binary), "ebcdic", and "local byte size" (for PDP-10's and PDP-20's mostly). *Ftp* supports the ascii and image types of file transfer.

Ftp supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

OPTIONS

Options may be specified at the command line, or to the command interpreter.

The **-v** (verbose on) option forces *ftp* to show all responses from the remote server, as well as report on data transfer statistics.

The **-n** option restrains *ftp* from attempting "auto-login" upon initial connection. If auto-login is enabled, *ftp* will check the *.netrc* file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* will use the login name on the local machine as the user identity on the remote machine, and prompt for a password and, optionally, an account with which to login.

The **-i** option turns off interactive prompting during mutliple file transfers.

The **-d** option enables debugging.

The **-g** option disables file name globbing.

BUGS

Many FTP server implementation do not support the experimental operations such as print working directory. Aborting a file transfer does not work right; if one attempts this the local *ftp* will likely have to be killed by hand.

NAME

gcore – get core images of running processes

SYNOPSIS

gcore process-id ...

DESCRIPTION

Gcore creates a core image of each specified process, suitable for use with *adb*(1) or *dbx*(1).

FILES

core.<process-id> core images

BUGS

Paging activity that occurs while *gcore* is running may cause the program to become confused. For best results, the desired processes should be stopped.

NAME

gprof − display call graph profile data

SYNOPSIS

gprof [options] [a.out [gmon.out ...]]

DESCRIPTION

gprof produces an execution profile of C, Pascal, or Fortran77 programs. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file (*gmon.out* default) which is created by programs which are compiled with the **−pg** option of *cc*, *pc*, and *f77*. That option also links in versions of the library routines which are compiled for profiling. The symbol table in the named object file (*a.out* default) is read and correlated with the call graph profile file. If more than one profile file is specified, the *gprof* output shows the sum of the profile information in the given profile files.

First, a flat profile is given, similar to that provided by *prof*(1). This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time.

Next, these times are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. A second listing shows the functions sorted according to the time they represent including the time of their call graph descendents. Below each function entry is shown its (direct) call graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendents is propagated to its (direct) call graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

The following options are available:

−a suppresses the printing of statically declared functions. If this option is given, all relevant information about the static function (*e.g.*, time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the *a.out* file.

−b supresses the printing of a description of each field in the profile.

−c the static call graph of the program is discovered by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.

−e *name*
suppresses the printing of the graph profile entry for routine *name* and all its descendants (unless they have other ancestors that aren't suppressed). More than one **−e** option may be given. Only one *name* may be given with each **−e** option.

−E *name*
suppresses the printing of the graph profile entry for routine *name* (and its descendants) as **−e**, above, and also excludes the time spent in *name* (and its descendants) from the total and percentage time computations. (For example, **−E** *mcount* **−E** *mcleanup* is the default.)

−f *name*
prints the graph profile entry of only the specified routine *name* and its descendants. More than one **−f** option may be given. Only one *name* may be given with each **−f** option.

−F *name*
prints the graph profile entry of only the routine *name* and its descendants (as **−f**, above) and also uses only the times of the printed routines in total time and percentage

computations. More than one **-F** option may be given. Only one *name* may be given with each **-F** option. The **-F** option overrides the **-E** option.

-s a profile file *gmon.sum* is produced which represents the sum of the profile information in all the specified profile files. This summary profile file may be given to subsequent executions of gprof (probably also with a **-s**) to accumulate profile data across several runs of an *a.out* file.

-z displays routines which have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the **-c** option for discovering which routines were never called.

FILES

a.out	the namelist and text space.
gmon.out	dynamic call graph and profile.
gmon.sum	summarized dynamic call graph and profile.

SEE ALSO

monitor(3), profil(2), cc(1), prof(1)
"gprof: A Call Graph Execution Profiler", by Graham, S.L., Kessler, P.B., McKusick, M.K.; *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

BUGS

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. We assume that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

Parents which are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

The profiled program must call *exit*(2) or return normally for the profiling information to be saved in the gmon.out file.

NAME

graph − draw a graph

SYNOPSIS

graph [option] ...

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 *plot*(1G) filters.

If the coordinates of a point are followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes "...", in which case they may be empty or contain blanks and numbers; labels never contain newlines.

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 (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **−x**).

−b Break (disconnect) 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.

−s Save screen, don't 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 Transpose horizontal and vertical axes. (Option **−x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **−s** option is present.

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

spline(1G), plot(1G)

BUGS

Graph stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

grep, egrep, fgrep − search a file for a pattern

SYNOPSIS

grep [option] ... expression [file] ...

egrep [option] ... [expression] [file] ...

fgrep [option] ... [strings] [file]

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular expressions in the style of *ex*(1); it uses a compact nondeterministic algorithm. *Egrep* patterns are full regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed strings; it is fast and compact. The following options are recognized.

−v All lines but those matching are printed.

−x (Exact) only lines matched in their entirety are printed (*fgrep* only).

−c Only a count of matching lines is printed.

−l The names of files with matching lines are listed (once) separated by newlines.

−n Each line is preceded by its relative line number in the file.

−b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

−i The case of letters is ignored in making comparisons − that is, upper and lower case are considered identical. This applies to *grep* and *fgrep* only.

−s Silent mode. Nothing is printed (except error messages). This is useful for checking the error status.

−w The expression is searched for as a word (as if surrounded by '\<' and '\>', see *ex*(1).) (*grep* only)

−e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a −.

−f *file* The regular expression (*egrep*) or string list (*fgrep*) is taken from the *file*.

In all cases the file name is shown if there is more than one input file. Care should be taken when using the characters $ * [^ | ( ) and \ in the *expression* as they are also meaningful to the Shell. It is safest to enclose the entire *expression* argument in single quotes ' '.

Fgrep searches for lines that contain one of the (newline-separated) *strings*.

Egrep accepts extended regular expressions. In the following description 'character' excludes newline:

A \ followed by a single character other than newline matches that character.

The character ^ matches the beginning of a line.

The character $ matches the end of a line.

A . (period) matches any character.

A single character not otherwise endowed with special meaning matches that character.

A string enclosed in brackets [] matches any single character from the string. Ranges of ASCII character codes may be abbreviated as in 'a−z0−9'. A] may occur only as the first character of the string. A literal − must be placed where it can't be mistaken

as a range indicator.

A regular expression followed by an * (asterisk) matches a sequence of 0 or more matches of the regular expression. A regular expression followed by a + (plus) matches a sequence of 1 or more matches of the regular expression. A regular expression followed by a ? (question mark) matches a sequence of 0 or 1 matches of the regular expression.

Two regular expressions concatenated match a match of the first followed by a match of the second.

Two regular expressions separated by | or newline match either a match for the first or a match for the second.

A regular expression enclosed in parentheses matches a match for the regular expression.

The order of precedence of operators at the same parenthesis level is [] then *+? then concatenation then | and newline.

Ideally there should be only one *grep*, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.

SEE ALSO

ex(1), sed(1), sh(1)

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

BUGS

Lines are limited to 256 characters; longer lines are truncated.

NAME

gripe — mail a local system bug report

SYNOPSIS

gripe [address]

DESCRIPTION

Bug reports sent to 'gripes' are intercepted by a program which expects bug reports to conform to a standard format. *Gripe* is a shell script to help the user compose and mail bug reports in the correct format. *Gripe* works by invoking *emacs*(1) (by default, unless the environment variable EDITOR is set) on a temporary copy of the bug report format outline. The user must fill in the appropriate fields and exit *emacs*. *Gripe* then mails the completed report to 'gripes' or the address specified on the command line.

FILES

/usr/ucb/bugformat contains the bug report outline

SEE ALSO

emacs(1), sendbug(1), sendmail(8)

NAME

groups — show group memberships

SYNOPSIS

groups [user]

DESCRIPTION

The *groups* command shows the groups to which you or the optionally specified user belong. Each user belongs to a group specified in the password file */etc/passwd* and possibly to other groups as specified in the file */etc/group*. If you do not own a file but belong to the group which it is owned by then you are granted group access to the file.

When a new file is created it is given the group of the containing directory.

SEE ALSO

setgroups(2)

FILES

/etc/passwd, /etc/group

BUGS

More groups should be allowed.

NAME

head — give first few lines

SYNOPSIS

head [−count] [file ...]

DESCRIPTION

This filter gives the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted it defaults to 10.

SEE ALSO

tail(1)

NAME

host – print IP host names and addresses

SYNOPSIS

host

host hostname | internet-address ...

DESCRIPTION

Host invoked with a host name or address prints the name(s) of that host and the primary internet address of the host.

Invoked with no arguments, it prints the names and address of the local host.

AUTHOR

Jeff Mogul

BUGS

This inherits lots of bugs from the library routines that look up hosts names.

It doesn't print multiple addresses for multi-homed hosts; its concept of "primary address" is arbitrary (but it matches the concept uses by most programs.)

NAME

hostid — set or print identifier of current host system

SYNOPSIS

hostid [identifier]

DESCRIPTION

The *hostid* command prints the identifier of the current host. This numeric value is expected to be unique across all hosts and is normally set to the host's Internet address. The super-user can set the hostid by giving an argument; this is usually done in the startup script /etc/rc.local.

SEE ALSO

gethostid(2), sethostid(2)

NAME

hostname – set or print name of current host system

SYNOPSIS

hostname [nameofhost]

DESCRIPTION

The *hostname* command prints the name of the current host, as given before the "login" prompt. The super-user can set the hostname by giving an argument; this is usually done in the startup script /etc/rc.local.

SEE ALSO

gethostname(2), sethostname(2)

NAME

iconc – compile and link Icon programs

SYNOPSIS

iconc [option ...] file ...

DESCRIPTION

Iconc is a compiler for Version 5 of the Icon programming language. Compilation consists of four phases: *translation*, *linking*, *assembling*, and *loading*. During translation, each Icon source file is translated into an intermediate language; during linking, the intermediate language files are combined and a single assembly code output file is produced which is then assembled; during loading, the assembled program is loaded with the Icon runtime system libraries, producing an executable file. Unless the **–o** option is specified, the name of the resulting executable file is formed by deleting the suffix of the first file named on the command line.

Files whose names end in '.icn' are assumed to be Icon source programs; they are translated, and the intermediate code is left in two files of the same name with '.u1' and '.u2' substituted for '.icn'. The intermediate code files are normally deleted when compilation has finished. Files whose names end in '.u1' or '.u2' are assumed to be intermediate code files from a previous translation (only one should be named – the other is assumed); these files are included in the linking phase after any '.icn' files have been translated. Files whose names end in '.c' or '.o' are assumed to be external functions. Any '.c' file is compiled using *cc*(1) to produce a '.o' file. A '.u1', '.u2', '.c', or '.o' file that is explicitly named is not deleted.

The following options are recognized by *iconc*.

–c Suppress the linking and loading phases. The intermediate code files are not deleted.

–m
Preprocess each '.icn' source file with the *m4*(1) macro processor before translation.

–o *output*
Name the executable file *output*.

–s Suppress any informative messages from the translator and linker. Normally, both informative messages and error messages are sent to standard error output.

–t Arrange for **&trace** to have an initial value of –1 when the program is executed. Normally, **&trace** has an initial value of 0.

–u Issue warning messages for undeclared identifiers in the program. The warnings are issued during the linking phase.

When an Icon program is executed, a number of environment variables are examined to determine certain execution parameters. The values assigned to these variables should be numbers. The variables that affect execution and the interpretations of their values are as follows:

TRACE
Initialize the value of **&trace**. If this variable has a value, it overrides the translation-time **–t** option.

NBUFS
The number of i/o buffers to use for files. When a file is opened, it is

assigned an i/o buffer if one is available and the file is not a tty. If no buffer is available, the file is not buffered. **&input**, **&output**, and **&errout** are buffered if buffers are available. On VAX systems, ten buffers are allocated initially; on PDP-11 systems, five buffers are allocated initially.

NOERRBUF
If set, **&errout** is not buffered.

STRSIZE
The initial size of the string space, in bytes. The string space grows if necessary, but it never shrinks. On VAX systems, the string space is initially 51,200 bytes; on PDP-11 systems, 10,240 bytes initially.

HEAPSIZE
The initial size of the heap, in bytes. The heap grows if necessary, but it never shrinks. On VAX systems, the heap is initially 51,200 bytes; on PDP-11 systems, 10,240 bytes initially.

NSTACKS
The number of stacks initially available for co-expressions. On VAX systems, four stacks are initially allocated; on PDP-11 systems, two stacks are initially allocated. More are automatically allocated if needed. It is unwise to set *NSTACKS* to 1.

STKSIZE
The size of each co-expression stack, in words. On VAX systems, stacks are normally 2000 words; on PDP-11 systems, stacks are normally 1000 words.

PROFILE
Turn on execution profiling of the runtime system. The value of this variable specifies the sampling resolution, in words. If the value is zero, profiling is not done. When a profiled program finishes, a file named 'mon.out' is created containing the results of the profile. The program *prof*(1) can be used to examine the results. This produces a profile of the runtime system, not the user program.

FILES

mon.out	results of profiling
v5g/cmp/bin/utran	icon translator
v5g/cmp/bin/ulink	icon linker
v5g/cmp/bin/libi.a	icon runtime library

SEE ALSO

The Icon Programming Language, Ralph E. Griswold and Madge T. Griswold, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1983.

Installation and Maintenance Guide for Release 5g of Icon, Department of Computer Science, The University of Arizona, March 1983.

cc(1), icont(1), ld(1), m4(1), prof(1), monitor(3)

BUGS

Because of the way that co-expressions are implemented, there is a possibility that programs in which they are used may malfunction mysteriously.

Integer overflow on multiplication is not detected.

If the **-m** option is used, line numbers reported in error messages or tracing messages are from the file after, not before, preprocessing.

NAME

icont – translate Icon programs for interpretive execution

SYNOPSIS

icont [option ...] file ... [**–x** arg ...]

DESCRIPTION

Icont is a translator for Version 5 of the Icon programming language, which produces a file suitable for interpretation by the Icon interpreter. Translation consists of two phases: *translation* and *linking*. During translation, each Icon source file is translated into an intermediate language; during linking, the intermediate language files are combined and a single output file is produced. The output file from the linker is referred to as an *interpretable* file. Unless the **–o** option is specified, the name of the resulting interpretable file is formed by deleting the suffix of the first input file named on the command line. If the **–x** argument is used, the file is automatically executed by the interpreter and any arguments following the **–x** are passed as execution arguments to the Icon program itself.

Files whose names end in '.icn' are assumed to be Icon source programs; they are translated, and the intermediate code is left in two files of the same name with '.u1' and '.u2' substituted for '.icn'. The intermediate code files normally are deleted when compilation has finished. Files whose names end in '.u1' or '.u2' are assumed to be intermediate code files from a previous translation (only one should be named – the other is assumed); these files are included in the linking phase after any '.icn' files have been translated. A '.u1' or '.u2' file that is explicitly named is not deleted. Icon source programs may be read from standard input. The argument – signifies the use of standard input as a source file. In this case, the intermediate code is placed in 'stdin.u1' and 'stdin.u2' and the interpretable file is 'stdin'.

The following options are recognized by *icont*.

–c Suppress the linking phase. The intermediate code files are not deleted.

–m
Preprocess each '.icn' source file with the *m4*(1) macro processor before translation.

–o *output*
Name the interpretable file *output*.

–s Suppress any informative messages from the translator and linker. Normally, both informative messages and error messages are sent to standard error output.

–t Arrange for **&trace** to have an initial value of –1 when the program is executed. Normally, **&trace** has an initial value of 0.

–u Issue warning messages for undeclared identifiers in the program. The warnings are issued during the linking phase.

The interpretable file produced by the Icon linker is *directly executable*. For example, the command

icont hello.icn

produces a file named **hello** that can be run by the command

hello

The method used to make interpretable files appear to be directly executable is system dependent. See the Icon installation guide for complete details. For most intents and purposes, interpretable files are executable programs in the same sense that files produced by *ld*(1) are executable programs.

Arguments can be passed to the Icon program by following the program name with the arguments. Any such arguments are passed to the main procedure as a list of strings.

When an Icon program is executed, a number of environment variables are examined to determine certain execution parameters. The values assigned to these variables should be numbers. The variables that affect execution and the interpretations of their values are as follows:

TRACE
Initialize the value of **&trace**. If this variable has a value, it overrides the translation-time **-t** option.

NBUFS
The number of i/o buffers to use for files. When a file is opened, it is assigned an i/o buffer if one is available and the file is not a tty. If no buffer is available, the file is not buffered. **&input**, **&output**, and **&errout** are buffered if buffers are available. On VAX systems, ten buffers are allocated initially; on PDP-11 systems, five buffers are allocated initially.

NOERRBUF
If set, **&errout** is not buffered.

STRSIZE
The initial size of the string space, in bytes. The string space grows if necessary, but it never shrinks. On VAX systems, the string space is initially 51,200 bytes; on PDP-11 systems, 10,240 bytes initially.

HEAPSIZE
The initial size of the heap, in bytes. The heap grows if necessary, but it never shrinks. On VAX systems, the heap is initially 51,200 bytes; on PDP-11 systems, 10,240 bytes initially.

NSTACKS
The number of stacks initially available for co-expressions. On VAX systems, four stacks are initially allocated; on PDP-11 systems, two stacks are initially allocated. More are automatically allocated if needed. It is unwise to set *NSTACKS* to 1.

STKSIZE
The size of each co-expression stack, in words. On VAX systems, stacks are normally 2000 words; on PDP-11 systems, stacks are normally 1000 words.

PROFILE
Turn on execution profiling of the runtime system. The value of this variable specifies the sampling resolution, in words. If the value is zero, profiling is not done. When a profiled program finishes, a file named 'mon.out' is created containing the results of the profile. The program *prof*(1) can be used to examine the results. This produces a profile of the runtime system, not the user program.

FILES

v5g/int/bin/utran	icon translator
v5g/int/bin/ulink	icon linker
v5g/int/bin/iconx	icon interpreter
mon.out	results of profiling

SEE ALSO

The Icon Programming Language, Ralph E. Griswold and Madge T. Griswold, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1983.

Installation and Maintenance Guide for Release 5g of Icon, Department of Computer Science, The University of Arizona, March 1983.

iconc(1), m4(1), prof(1), monitor(3)

BUGS

Downward compatibility of interpretable files will not be maintained in subsequent releases of Icon. No checks are performed to determine if the interpretable file and the interpreter are compatible. Peculiar program behavior is the only indication of such incompatibility.

Interpretable files do not stand alone; the Icon interpreter must be present on the system. This implies that an interpretable file produced on one system will not work on another system unless the Icon interpreter is in the same place on both systems and that the interpreter is of the same version of Icon as the translator that produced the interpretable file.

Because of the way that co-expressions are implemented, there is a possibility that programs in which they are used may malfunction mysteriously.

Integer overflow on multiplication is not detected.

If the **-m** option is used, line numbers reported in error messages or tracing messages are from the file after, not before, preprocessing.

NAME

ident − identify files

SYNOPSIS

ident file ...

DESCRIPTION

Ident searches the named files for all occurrences of the pattern *$keyword:...$*, where *keyword* is one of

Author
Date
Header
Locker
Log
Revision
Source
State

These patterns are normally inserted automatically by the RCS command *co (1)*, but can also be inserted manually.

Ident works on text files as well as object files. For example, if the C program in file f.c contains

```
char rcsid[] = "$Header: Header information $";
```

and f.c is compiled into f.o, then the command

```
ident f.c f.o
```

will print

```
f.c:
     $Header: Header information $
f.o:
     $Header: Header information $
```

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.0 ; Release Date: 82/12/04 .

SEE ALSO

ci (1), co (1), rcs (1), rcsdiff(1), rcsintro (1), rcsmerge (1), rlog (1), rcsfile (5).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

BUGS

NAME

imprint - print text files on Imprint-10

SYNOPSIS

imprint [*pr* options] [options] [file...]

DESCRIPTION

imprint calls *pr* or *cat* on the input files and prefixes an appropriate document header. If no file names are given, the standard input is used. The user must manually pipe the output of imprint into the spooler on his system (e.g. "imprint -2 file | lpr").

Imprint should only be used for printers that understand the new document header (V1.0 and later). Earlier printers must use the *iprint* program, which first converts the text file to DVI format.

Options for *imprint* fall into several catagories. All options not listed below are regarded as options to *pr* or *cat* and are passed to it. Certain of the options listed below are also options to *pr* and are passed to it as well as having other effects. The secondary effect of these options can be suppressed by preceeding them with the **−P** switch.

The following flags are accepted:

−f Fold long lines, instead of truncating them.

−N Prefix a line number to each line; also turns on **−f** fold switch (see above).

−P The next argument is to be passed to *pr* or *cat*.

−c*n* Print *n* copies. For V1.0 and later systems this turns on pagecollation.

−h The following argument is used as the banner for *pr* and as the file name for the header page of the job (for V1.0 and later systems only).

−l*n* Set the page length to *n* lines. For V1.0 and later systems this may also set the printer's interline spacing.

−n Use *cat* rather than *pr* to print the file.

−o*n* Print with a page offset (left margin) of *n* spaces (this is done in the printer (or *iprint*); this argument is not passed to *pr*).

−w*n* Set the the line width to *n* characters. For V1.0 and later systems, a line width of more than 80 characters is printed in landscape (132 column) mode.

−2 Print two logical pages per physical page (2 up).

−C Suppress pagecollation (see **−c** above).

−F Suppress pagereversal (which is on by default).

−J Suppress generation of the job header page.

−L Print in landscape mode, 132 columns wide.

−O Print page outlines.

−R Print page rules (one every two lines).

SEE ALSO

pr(1), cat(1)

DIAGNOSTICS

See diagnostic messages for *pr* and *cat*.

BUGS

For V1.0 and later systems, certain parameters can be overriden by document control language strings in the file itself. Also, a **−c***n* flag after a **−C** flag turns pagecollation on once more.

If the job contains errors detected by the printer, the job header page will always be generated.

HISTORY

2-May-1983 Jan Stoeckenius, Imagen, from an earlier shell file written by Steve Tepper.
26-Jan-1984 Bill Croft, Stanford; added **−f** and **−N** switches.

NAME

inc – incorporate new mail

SYNOPSIS

inc [+folder] [–audit audit-file] [–help]

DESCRIPTION

Inc incorporates mail from the user's incoming mail drop (mail) into an MH folder. If '+folder' isn't specified, the folder named "inbox" in the user's MH directory will be used. The new messages being incorporated are assigned numbers starting with the next highest number in the folder. If the specified (or default) folder doesn't exist, the user will be queried prior to its creation. As the messages are processed, a *scan* listing of the new mail is produced.

If the user's profile contains a "Msg–Protect: nnn" entry, it will be used as the protection on the newly created messages, otherwise the MH default of 664 will be used. During all operations on messages, this initially assigned protection will be preserved for each message, so *chmod*(1) may be used to set a protection on an individual message, and its protection will be preserved thereafter.

If the switch '–audit audit-file' is specified (usually as a default switch in the profile), then *inc* will append a header line and a line per message to the end of the specified audit-file with the format:

```
inc date
        <scan line for first message>
        <scan line for second message>
                <etc.>
```

This is useful for keeping track of volume and source of incoming mail. Eventually, *repl*, *forw*, *comp*, and *dist* may also produce audits to this (or another) file, perhaps with "Message-Id:" information to keep an exact correspondence history. "Audit-file" will be in the user's MH directory unless a full path is specified.

Inc will incorporate even illegally formatted messages into the user's MH folder, inserting a blank line prior to the offending component and printing a comment identifying the bad message.

In all cases, the mail file will be zeroed.

FILES

$HOME/mh_profile	The user profile
$HOME/mail	The user's mail drop
<mh-dir>/audit-file	Audit trace file (optional)

PROFILE COMPONENTS

Path:	To determine the user's MH directory
Folder–Protect:	For protection on new folders
Msg–Protect:	For protection on new messages

DEFAULTS

'+folder' defaults to "inbox"

CONTEXT

The folder into which the message is being incorporated will become the current folder, and the first message incorporated will be the current message. This leaves the context ready for a *show* of the first new message.

NAME

include – search for and print header (include) files

SYNOPSIS

include [-acnq.] [files ...]

DESCRIPTION

Include is used to find and print the contents of header (".h") files. There are a number of places where a header file might be found; *include* has a list of several places to search.

Invoking *include* with no **file** arguments causes it to print a list of the directories searched; these are searched in the order they are printed here. Your home directory and its "include/" subdirectory are included. It is not an error if some of the listed directories do not exist (or are not searchable.) The directory "/" is a special case; this means that an absolute pathname is expected.

Invoking *include* with one or more **file** arguments causes it to search for the specified file in the directories on the search list. In each directory, *include* first looks for **file** and, if that is not there, **file.h**. If found, the file is printed on the terminal, using *more*(1).

FLAGS

–a	Normally, *include* searches for a file only until it finds it. This flag causes *include* to look for as many instances of the file as it can.
–c	Tells *include* to use *cat*(1) instead of *more*(1) to print the file on the terminal.
–n	Tells *include* to print only the names of the files found; the contents are not printed.
–q	Tells *include* not to print the names of the files; only the contents are printed (unless the **–n** flag is on.)
–.	Tells *include* not to search "./" (i.e., the current working directory) as such; it may be searched if it is also named by another directory on the search list.

DIAGNOSTICS

Indicates if a file is not found anywhere.

AUTHOR

Jeffrey Mogul

BUGS

NAME

indent − indent and format C program source

SYNOPSIS

indent *input* [*output*] [*flags*]

DESCRIPTION

Indent is intended primarily as a C program formatter. Specifically, *indent* will:

- indent code lines
- align comments
- insert spaces around operators where necessary
- break up declaration lists as in "int a,b,c;".

Indent will not break up long statements to make them fit within the maximum line length, but it will flag lines that are too long. Lines will be broken so that each statement starts a new line, and braces will appear alone on a line. (See the −br option to inhibit this.) Also, an attempt is made to line up identifiers in declarations.

The *flags* which can be specified follow. They may appear before or after the file names. If the *output* file is omitted, the formatted file will be written back into *input* and a "backup" copy of *input* will be written in the current directory. If *input* is named "/blah/blah/file", the backup file will be named ".Bfile". If *output* is specified, *indent* checks to make sure it is different from *input*.

The following flags may be used to control the formatting style imposed by *indent*.

−l*nnn* Maximum length of an output line. The default is 75.

−c*nnn* The column in which comments will start. The default is 33.

−cd*nnn* The column in which comments on declarations will start. The default is for these comments to start in the same column as other comments.

−i*nnn* The number of spaces for one indentation level. The default is 4.

−dj, −ndj **−dj** will cause declarations to be left justified. **−ndj** will cause them to be indented the same as code. The default is **−ndj**.

−v, −nv **−v turns on "verbose" mode, −nv** turns it off. When in verbose mode, *indent* will report when it splits one line of input into two or more lines of output, and it will give some size statistics at completion. The default is **−nv**.

−bc, −nbc
If **−bc** is specified, then a newline will be forced after each comma in a declaration. **−nbc** will turn off this option. The default is **−bc**.

−d*nnn* This option controls the placement of comments which are not to the right of code. Specifying **−d2** means that such comments will be placed two indentation levels to the left of code. The default **−d0** lines up these comments with the code. See the section on comment indentation below.

−br, −bl Specifying **−bl** will cause complex statements to be lined up like this:

```
if (...)
{
    code
}
```

Specifying **−br** (the default) will make them look like this:

```
if (...) {
    code
}
```

You may set up your own "profile" of defaults to *indent* by creating the file ".indent.pro" in your login directory and including whatever switches you like. If *indent* is run and a profile file exists, then it is read to set up the program's defaults. Switches on the command line, though, will always override profile switches. The profile file must be a single line of not more than 127 characters. The switches should be separated on the line by spaces or tabs.

Multi-line expressions

Indent will not break up complicated expressions that extend over multiple lines, but it will usually correctly indent such expressions which have already been broken up. Such an expression might end up looking like this:

```
x =
    (
        (Arbitrary parenthesized expression)
        +
        (
            (Parenthesized expression)
            *
            (Parenthesized expression)
        )
    );
```

Comments

Indent recognizes four kinds of comments. They are: straight text, "box" comments, UNIX-style comments, and comments that should be passed through unchanged. The action taken with these various types are as follows:

"Box" comments. *Indent* assumes that any comment with a dash immediately after the start of comment (i.e. "/*—") is a comment surrounded by a box of stars. Each line of such a comment will be left unchanged, except that the first non-blank character of each successive line will be lined up with the beginning slash of the first line. Box comments will be indented (see below).

"Unix-style" comments. This is the type of section header which is used extensively in the UNIX system source. If the start of comment ("/*") appears on a line by itself, indent assumes that it is a UNIX-style comment. These will be treated similarly to box comments, except the first non-blank character on each line will be lined up with the '*' of the "/*".

Unchanged comments. Any comment which starts in column 1 will be left completely unchanged. This is intended primarily for documentation header pages. The check for unchanged comments is made before the check for UNIX-style comments.

Straight text. All other comments are treated as straight text. *Indent* will fit as many words (separated by blanks, tabs, or newlines) on a line as possible. Straight text comments will be indented.

Comment indentation

Box, UNIX-style, and straight text comments may be indented. If a comment is on a line with code it will be started in the "comment column", which is set by the **−cnnn** command line parameter. Otherwise, the comment will be started at *nnn* indentation levels less than where code is currently being placed, where *nnn* is specified by the **−dnnn** command line parameter. (Indented comments will never be placed in column 1.) If the code on a line extends past the comment column, the comment will be moved to the next line.

DIAGNOSTICS

Diagnostic error messages, mostly to tell that a text line has been broken or is too long for the output line.

FILES

.indent.pro profile file

BUGS

Does not know how to format "long" declarations.

NAME

inews – submit news articles

SYNOPSIS

inews [**–h**] **–t** *title* [**–n** *newsgroups*] [**–e** *expiration date*]

inews –p [*filename*]

inews –C *newsgroup*

DESCRIPTION

Inews submits news articles to the USENET news network. It is intended as a raw interface, not as a human user interface. Casual users should probably use *postnews*(1) instead.

The first form is for submitting user articles. The body will be read from the standard input. A *title* must be specified as there is no default. Each article belongs to a list of newsgroups. If the **–n** flag is omitted, the list will default to something line *general*. (On ours, it is *general*.) If you wish to submit an article in multiple newsgroups, the *newsgroups* must be separated by commas and/or spaces. If not specified, the expiration date will be set to the local default. The **–f** flag specifies the article's sender. Without this flag, the sender defaults to the user's name. If **–f** is specified, the real sender's name will be included as a Sender line. The **–h** flag specifies that headers are present at the beginning of the article, and these headers should be included with the article header instead of as text. (This mechanism can be used to edit headers and supply additional nondefault headers, but not to specify certain information, such as the sender and article ID, that inews itself generates.)

When posting an article, the environment is checked for information about the sender. If NAME is found, its value is used for the full name, rather than the system value (often in /etc/passwd). This is useful if the system value cannot be set, or when more than one person uses the same login. If ORGANIZATION is found, the value overrides the system default organization. This is useful when a person uses a guest login and is not primarily associated with the organization owning the machine.

The second form is used for receiving articles from other machines. If *filename* is given, the article will be read from the specified file; otherwise the article will be read from the standard input. An expiration date need not be present and a receival date, if present, will be ignored.

After local installation, inews will transmit the article to all systems that subscribe to the newsgroups that the article belongs to.

The third form is for creating new newsgroups. On some systems, this may be limited to specific users such as the super-user or news administrator. (This happens on ours.)

If the file /usr/lib/news/recording is present, it is taken as a list of "recordings" to be shown to users posting news. (This is by analogy to the recording you hear when you dial information in some parts of the country, asking you if you really wanted to do this.) The file contains lines of the form:

newsgroups <tab> filename

for example:

net.all net.recording fa.all fa.recording

Any user posting an article to a newsgroup matching the pattern on the left will be shown the contents of the file on the right. The file is found in the LIB directory (often /usr/lib/news). The user is then told to hit DEL to abort or RETURN

to proceed. The intent of this feature is to help companies keep proprietary information from accidently leaking out.

FILES

/usr/spool/news/.sys.nnn	Temporary articles
/usr/spool/news/*newsgroups/article_no.*	Articles
/usr/spool/oldnews/	Expired articles
/usr/lib/news/active	List of known newsgroups and highest local article numbers in each.
/usr/lib/news/seq	Sequence number of last article
/usr/lib/news/history	List of all articles ever seen
/usr/lib/news/sys	System subscription list

SEE ALSO

Mail(1), binmail(1), getdate(3), msgs(1), news(5), newsrc(5), postnews(1), readnews(1), recnews(1), sendnews(8), uucp(1), uurec(8),

AUTHORS

Matt Glickman
Mark Horton
Stephen Daniel
Tom R. Truscott

NAME

ingroup – show membership in a specified group

SYNOPSIS

ingroup group ...

DESCRIPTION

The *ingroup* command shows the users who belong to the specified group(s).

Each user belongs to a group specified in the password file */etc/passwd* and possibly to other groups as specified in the file */etc/group*.

SEE ALSO

groups(1)

FILES

/etc/passwd, /etc/group

BUGS

Probably should be a function of *groups*(1).

NAME

install − install binaries

SYNOPSIS

install [**−c**] [**−m** mode] [**−o** owner] [**−g** group] [**−s**] binary destination

DESCRIPTION

Binary is moved (or copied if **−c** is specified) to *destination.* If *destination* already exists, it is removed before *binary* is moved. If the destination is a directory then *binary* is moved into the *destination* directory with its original file-name.

The mode for *Destination* is set to 755; the **−m** *mode* option may be used to specify a different mode.

Destination is changed to owner root; the **−o** *owner* option may be used to specify a different owner.

Destination is changed to group staff; the **−g** *group* option may be used to specify a different group.

If the **−s** option is specified the binary is stripped after being installed.

Install refuses to move a file onto itself.

SEE ALSO

chgrp(1), chmod(1), cp(1), mv(1), strip(1), chown(8)

NAME

iostat − report I/O statistics

SYNOPSIS

iostat [interval [count]]

DESCRIPTION

Iostat iteratively reports the number of characters read and written to terminals, and, for each disk, the number of seeks transfers per second, kilobytes transfered per second, and the milliseconds per average seek. It also gives the percentage of time the system has spent in user mode, in user mode running low priority (niced) processes, in system mode, and idling.

To compute this information, for each disk, seeks and data transfer completions and number of words transferred are counted; for terminals collectively, the number of input and output characters are counted. Also, each sixtieth of a second, the state of each disk is examined and a tally is made if the disk is active. From these numbers and given the transfer rates of the devices it is possible to determine average seek times for each device.

The optional *interval* argument causes *iostat* to report once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports.

FILES

/dev/kmem
/vmunix

SEE ALSO

vmstat(1)

NAME

iphostid – set or print Internet Protocol (IP) identifier of current host

SYNOPSIS

hostid [identifier]

DESCRIPTION

The *iphostid* command prints the identifier of the current host in both hexadecimal and Internet (A.B.C.D) notations. This value is expected to be unique across all hosts and is normally set to the host's Internet address. The super-user can set the hostid by giving a host name or number (A.B.C.D) argument; this is usually done in the startup script /etc/rc.local.

This command is essentially identical to *hostid*, except that it does not require one to use or understand hexadecimal notation.

SEE ALSO

hostid(1), gethostid(2), sethostid(2)

NAME

iprint - convert text files to DVI format

SYNOPSIS

iprint [options] [file...]

DESCRIPTION

iprint converts the input text *files* to DVI format for printing on an Imprint-10 printer. If no file names are given, the standard input is used.

−i Use the next argument as the name of the output file.

−b Print the next argument on the banner page.

−c*n* Print *n* copies.

−B Print the first non-blank line on each page in a bold type face, and ignore leading blank lines. This is for use with programs like *pr* which generate page headers.

−f Use the following argument as the name of a font file for the text. A variable-pitch font will generally produce ugly results.

−F Use the following argument as the name of a font file for the bold header line (see description of the **−B** flag).

−o*n* Print with a page offset (left margin) of *n* spaces.

−l*n* Take the page length to be *n* lines.

If the **−i** flag is not used, the output is written into a temporary file and *dviimp* is called to process it.

FILES

/tmp/dvi??????
/usr/local/fonts/imagen/raster/*

SEE ALSO

imprint(1), dviimp(1), ipr(1)

DIAGNOSTICS

'Font *f* version *n*' means this font file is not a version 0 RAS-format file. Other diagnostics should be self-explanatory.

Specifying the **−v** flag will produce somewhat more verbose output. Specifying the **−d** flag will produce extensive debugging output.

AUTHOR

Imagen Corp.

NAME

itroff – troff to the ImPrint printer

SYNOPSIS

itroff [*troff* options **] [** flags **] [** *file*... **]**

DESCRIPTION

Itroff runs *troff*(1) in an environment to produce typeset output on the ImPrint-10 printer. It uses the *catdvi* program to convert to DVI format.

Besides the *troff* flags, the following are recognized:

−c*n* Print *n* copies.

−M*n* Set the Imprint-10 memory parameter to *n*, where $1 \leq n \leq 5$. See the Imprint-10 Programmer's Guide for a description of this value. This parameter applies only for pre-release systems (RELEASE = 0).

SEE ALSO

imprint(1), ipq(1), ipr(1), iprm(1), troff(1)

BUGS

Not all characters are handled optimally. Italic spacing tends to be a little tight. Cut marks fall just outside the page boundaries, thus the first set of cut marks is lost and all subsequent ones fall at the bottom of the preceeding page. Backwards motion across page boundaries, as is sometimes done in *tbl*, will cause errors.

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.

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*.

Fields are normally separated by blank, tab or newline. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

−a*n* In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.

−e *s* Replace empty output fields by string *s*.

−j*n m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file.

−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.

−t*c* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant.

SEE ALSO

sort(1), comm(1), awk(1)

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, look* and *awk*(1) are wildly incongruous.

NAME

kill − terminate a process with extreme prejudice

SYNOPSIS

kill [−sig] processid ...
kill −l

DESCRIPTION

Kill sends the TERM (terminate, 15) signal to the specified processes. If a signal name or number preceded by '−' is given as first argument, that signal is sent instead of terminate (see *sigvec*(2)). The signal names are listed by 'kill −l', and are as given in */usr/include/signal.h*, stripped of the common SIG prefix.

The terminate signal will kill processes that do not catch the signal; 'kill −9 ...' is a sure kill, as the KILL (9) signal cannot be caught. By convention, if process number 0 is specified, all members in the process group (i.e. processes resulting from the current login) are signaled (but beware: this works only if you use *sh*(1); not if you use *csh*(1).) The killed processes must belong to the current user unless he is the super-user.

The process number of an asynchronous process started with '&' is reported by the shell. Process numbers can also be found by using *Kill* is a built-in to *csh*(1); it allows job specifiers "%..." so process id's are not as often used as *kill* arguments. See *csh*(1) for details.

SEE ALSO

csh(1), ps(1), kill(2), sigvec(2)

BUGS

An option to kill process groups ala *killpg*(2) should be provided; a replacement for "kill 0" for *csh*(1) users should be provided.

NAME

last − indicate last logins of users and teletypes

SYNOPSIS

last [−N] [name ...] [tty ...]

DESCRIPTION

Last will look back in the *wtmp* file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example 'last 0' is the same as 'last tty0'. If multiple arguments are given, the information which applies to any of the arguments is printed. For example 'last root console' would list all of "root's" sessions as well as all sessions on the console terminal. *Last* will print the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, *last* so indicates.

The pseudo-user **reboot** logs in at reboots of the system, thus

```
last reboot
```

will give an indication of mean time between reboot.

Last with no arguments prints a record of all logins and logouts, in reverse order. The −N option limits the report to N lines.

If *last* is interrupted, it indicates how far the search has progressed in *wtmp*. If interrupted with a quit signal (generated by a control-\) *last* indicates how far the search has progressed so far, and the search continues.

FILES

/usr/adm/wtmp	login data base
/usr/adm/shutdownlog	which records shutdowns and reasons for same

SEE ALSO

wtmp(5), ac(8), lastcomm(1)

AUTHOR

Howard Katseff

NAME

lastcomm − show last commands executed in reverse order

SYNOPSIS

lastcomm [command name] ... [user name] ... [terminal name] ...

DESCRIPTION

Lastcomm gives information on previously executed commands. With no arguments, *lastcomm* prints information about all the commands recorded during the current accounting file's lifetime. If called with arguments, only accounting entries with a matching command name, user name, or terminal name are printed. So, for example,

lastcomm a.out root ttyd0

would produce a listing of all the executions of commands named *a.out* by user *root* on the terminal *ttyd0*.

For each process entry, the following are printed.

The name of the user who ran the process.
Flags, as accumulated by the accounting facilities in the system.
The command name under which the process was called.
The amount of cpu time used by the process (in seconds).
The time the process exited.

The flags are encoded as follows: "S" indicates the command was executed by the super-user, "F" indicates the command ran after a fork, but without a following *exec*, "C" indicates the command was run in PDP-11 compatibility mode (VAX only), "D" indicates the command terminated with the generation of a *core* file, and "X" indicates the command was terminated with the signal SIGTERM.

SEE ALSO

last(1), sigvec(2), acct(5), core(5)

NAME

latex – TeX with a macro package preloaded

SYNOPSIS

latex [first line]

DESCRIPTION

LaTeX is a set of TeX macros that provides the user with a complete document-preparation system. LaTeX was inspired by Scribe, but makes no effort to emulate it. In addition to the capabilities of PLAIN TeX, LaTeX provides many features, including the following:

- Automatic generation of section numbers and table of contents.
- Various forms of list-making commands, like enumerated lists.
- Commands to generate references to page and section numbers from symbolic labels.
- Automatic numbering of and symbolic referencing to bibliography citations. (Under construction is an auxiliary program to get bibliography entries from a central bibliographic database using the citations in the text.)
- Floating of figures and tables using a very sophisticated placement algorithm.
- Commands for drawing pictures with lines and arrows (horizontal, vertical and slanted), circles and quarter circles.
- Very convenient commands for generating arrays and tabular layout of text.
- Commands that allow you to run only part of your document through TeX with all the page and section numbering coming out right.

According to LaTeX's designer, Leslie Lamport, LaTeX is not simply a collection of nifty macros. It is an integrated SYSTEM for producing documents. It provides the user with a coherent model --- a much simpler and more coherent model than TeX presents --- which behaves consistently. (For example, enumerated lists do the right thing when nested within one another, or when they contain or appear inside other commands.)

The following "document styles" are currently installed: article (short documents), report (longer ones, with chapters), suthesis (Stanford PhD theses), and letter (for letters). Document style options are 11pt and 12pt (to change the "normal" point side), and twocolumn. The "normal" page layout style described in the LaTeX manual is available, but it wastes a lot of space on Dover output. The use of the "fullpage" page layout style is recommended.

There is also a program called *slitex* for making slides. Run latex and input "slides" to get information about how to use it.

More information on LateX can be found in the user's manual, "The LaTeX Document Preparation System." Also consult lerrata.tex (obtained by running "latex lerrata.tex") for revisions to the current manual. The manual entry for *tex*(1) should also be consulted for general information on running LaTeX.

SEE ALSO

Leslie Lamport, *The LaTeX Document Preparation System.*
tex(1)
dvip(1)

AUTHORS

LaTeX was designed and implemented by Leslie Lamport. TeX was designed by Donald E. Knuth, who implemented it using his WEB system for Pascal programs. Installed at Stanford by Howard Trickey.

NAME

ld − link editor

SYNOPSIS

ld [option] ... file ...

DESCRIPTION

Ld combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object *files* are given, and *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the **−r** option must be given to preserve the relocation bits.) The output of *ld* is left on **a.out**. This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine (unless the **−e** option is specified).

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by *ranlib*(1), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. The first member of a library should be a file named '__.SYMDEF', which is understood to be a dictionary for the library as produced by *ranlib*(1); the dictionary is searched iteratively to satisfy as many references as possible.

The symbols '_etext', '_edata' and '_end' ('etext', 'edata' and 'end' in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several options. Except for **−l**, they should appear before the file names.

−A This option specifies incremental loading, i.e. linking is to be done in a manner so that the resulting object may be read into an already executing program. The next argument is the name of a file whose symbol table will be taken as a basis on which to define additional symbols. Only newly linked material will be entered into the text and data portions of **a.out**, but the new symbol table will reflect every symbol defined before and after the incremental load. This argument must appear before any other object file in the argument list. The **−T** option may be used as well, and will be taken to mean that the newly linked segment will commence at the corresponding address (which must be a multiple of 1024). The default value is the old value of _end.

−D Take the next argument as a hexadecimal number and pad the data segment with zero bytes to the indicated length.

−d Force definition of common storage even if the **−r** flag is present.

−e The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default.

−l*x* This option is an abbreviation for the library name '/lib/lib*x*.a', where *x* is a string. If that does not exist, *ld* tries '/usr/lib/lib*x*.a' A library is searched when its name is encountered, so the placement of a **−l** is significant.

−M produce a primitive load map, listing the names of the files which will be loaded.

−N Do not make the text portion read only or sharable. (Use "magic number" 0407.)

−n Arrange (by giving the output file a 0410 "magic number") that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 1024 byte boundary following the end of the text.

-o The *name* argument after **-o** is used as the name of the *ld* output file, instead of **a.out**.

-r Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.

-S 'Strip' the output by removing all symbols except locals and globals.

-s 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debuggers). This information can also be removed by *strip*(1).

-T The next argument is a hexadecimal number which sets the text segment origin. The default origin is 0.

-t ("trace") Print the name of each file as it is processed.

-u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-X Save local symbols except for those whose names begin with 'L'. This option is used by *cc*(1) to discard internally-generated labels while retaining symbols local to routines.

-x Do not preserve local (non-.globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

-y*sym* Indicate each file in which *sym* appears, its type and whether the file defines or references it. Many such options may be given to trace many symbols. (It is usually necessary to begin *sym* with an '_', as external C, FORTRAN and Pascal variables begin with underscores.)

-z Arrange for the process to be loaded on demand from the resulting executable file (413 format) rather than preloaded. This is the default. Results in a 1024 byte header on the output file followed by a text and data segment each of which have size a multiple of 1024 bytes (being padded out with nulls in the file if necessary). With this format the first few BSS segment symbols may actually appear (from the output of *size*(1)) to live in the data segment; this to avoid wasting the space resulting from data segment size roundup.

FILES

/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
/usr/local/lib/lib*.a	still more libraries
a.out	output file

SEE ALSO

as(1), ar(1), cc(1), ranlib(1)

BUGS

There is no way to force data to be page aligned. *Ld* pads images which are to be demand loaded from the file system to the next page boundary to avoid a bug in the system.

NAME

ld68 – .b -> b.out linker for the MC68000

SYNOPSIS

ld68 [option] ... file ...

DESCRIPTION

Ld68 combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object *files* are given, and *ld68* combines them, producing an object module which can be either executed or become the input for a further *ld68* run. (In the latter case, the **–r** option must be given to preserve the relocation bits.) The output of *ld68* is left on **b.out**. This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified.

The entry point of the output is determined by the first applicable item of the following list: the **–e** option if given, the value of the symbol _start if defined, or the text origin (first instruction).

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important.

The symbols '_etext', '_edata' and '_end' ('etext', 'edata' and 'end' in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld68 understands several options. Except for **–l**, they should appear before the file names.

–D Take the next argument as a decimal number and pad the data segment with zero bytes to the indicated length.

–d Force definition of common storage even if the **–r** flag is present.

–e The following argument is taken to be the name of the entry point of the loaded program; location 0x1000 is the default.

–f Fold case on identifiers. That is, upper and lower case letters are not distinguished. Used to link with Pascal routines, for example.

–l*x* This option is an abbreviation for the library name '/usr/sun/lib/lib*x*.a', where *x* is a string. A library is searched when its name is encountered, so the placement of a **–l** is significant.

–v*x* This denotes board version *x* which may at present only be 'm' for Motorola Design Module. The default board version is the Sun1 prototype, v1.

–M Create a human-readable list of symbols in "sym.out".

–n Arrange (by giving the output file a 0410 "magic number") that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 64K byte boundary following the end of the text (not really useful yet).

–o The *name* argument after **–o** is used as the name of the *ld68* output file, instead of **b.out**.

–q Quicksort symbols in **b.out** in ascending numerical order.

–r Generate relocation bits in the output file so that it can be the subject of another *ld68* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.

–S 'Strip' the output by removing all symbols except locals and globals.

–s 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but

impair the usefulness of the debuggers). This information can also be removed by *strip*(1).

-T The next argument is a hexadecimal number which sets the text segment origin. The default origin is 0x1000. If you intend to use the output as input to another run of ld68, you must specify -T 0.

-B The next argument is a hexadecimal number which sets the common/bss segment origin. The default origin is immediately after the data segment.

-u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-X Save local symbols except for those whose names begin with 'L'. This option is used by *cc*(1) to discard internally-generated labels while retaining symbols local to routines.

-x Do not preserve local (non-.globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

FILES

/usr/sun/lib/lib*.a	libraries
b.out	output file

SEE ALSO

ar(1), cc68(1), a68(1)

BUGS

The b.out format header does not contain any indication of the text segment origin, so if you specify something other than the default origin -T 1000, you will have to remember this value and specify it again to dl68 when you download. The standard Sun monitor cannot netload files with origins other than 1000, so you must either use dl68 or write a special loader for such programs.

NAME

learn − computer aided instruction about UNIX

SYNOPSIS

learn [−directory] [subject [lesson]]

DESCRIPTION

Learn gives Computer Aided Instruction courses and practice in the use of UNIX, the C Shell, and the Berkeley text editors. To get started simply type **learn**. The program will ask questions to find out what you want to do. Some questions may be bypassed by naming a *subject*, and more yet by naming a *lesson*. You may enter the *lesson* as a number that *learn* gave you in a previous session. If you do not know the lesson number, you may enter the *lesson* as a word, and *learn* will look for the first lesson containing it. If the *lesson* is '−', *learn* prompts for each lesson; this is useful for debugging.

The *subject*'s presently handled are

files
editor
vi
morefiles
macros
eqn
C

There are a few special commands. The command 'bye' terminates a *learn* session and 'where' tells you of your progress, with 'where m' telling you more. The command 'again' re-displays the text of the lesson and 'again *lesson*' lets you review *lesson*.

The −*directory* option allows one to exercise a script in a nonstandard place.

FILES

/usr/lib/learn	subtree for all dependent directories and files
/usr/tmp/pl*	playpen directories

SEE ALSO

csh(1), ex(1)

BUGS

The main strength of *learn*, that it asks the student to use the real UNIX, also makes possible baffling mistakes. It is helpful, especially for nonprogrammers, to have a UNIX initiate near at hand during the first sessions.

Occasionally lessons are incorrect, sometimes because the local version of a command operates in a non-standard way. Such lessons may be skipped with the 'skip' command, but it takes some sophistication to recognize the situation.

To find a *lesson* given as a word, *learn* does a simple *fgrep*(1) through the lessons. It is unclear whether this sort of subject indexing is better than none.

Spawning a new shell is required for each of many user and internal functions.

NAME

leave − remind you when you have to leave

SYNOPSIS

leave [hhmm]

DESCRIPTION

Leave waits until the specified time, then reminds you that you have to leave. You are reminded 5 minutes and 1 minute before the actual time, at the time, and every minute thereafter. When you log off, *leave* exits just before it would have printed the next message.

The time of day is in the form hhmm where hh is a time in hours (on a 12 or 24 hour clock). All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If no argument is given, *leave* prompts with "When do you have to leave?". A reply of newline causes *leave* to exit, otherwise the reply is assumed to be a time. This form is suitable for inclusion in a *.login* or *.profile*.

Leave ignores interrupts, quits, and terminates. To get rid of it you should either log off or use "kill −9" giving its process id.

SEE ALSO

calendar(1)

AUTHOR

Mark Horton

BUGS

NAME

lex − generator of lexical analysis programs

SYNOPSIS

lex [−tvfn] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analyis of text. The input *files* (standard input default) contain regular expressions to be searched for, and actions written in C to be executed when expressions are found.

A C source program, 'lex.yy.c' is generated, to be compiled thus:

```
cc lex.yy.c −ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized.

The options have the following meanings.

−t Place the result on the standard output instead of in file "lex.yy.c".

−v Print a one-line summary of statistics of the generated analyzer.

−n Opposite of **−v**; **−n** is default.

−f "Faster" compilation: don't bother to pack the resulting tables; limited to small programs.

EXAMPLE

```
lex lexcommands
```

would draw *lex* instructions from the file *lexcommands,* and place the output in *lex.yy.c*

```
%%
[A−Z] putchar(yytext[0]+'a'−'A');
[ ]+$
[ ]+    putchar(' ');
```

is an example of a *lex* program that would be put into a *lex* command file. This program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

SEE ALSO

yacc(1), sed(1)

M. E. Lesk and E. Schmidt, *LEX − Lexical Analyzer Generator*

NAME

linelen – print line lengths for a text file

SYNOPSIS

linelen [–m] [filename]

DESCRIPTION

Linelen reads lines from the specified file (or the standard input, if no filename is given) and copies them to the standard output; each line is preceeded by its length. For example, "linelen /etc/fstab.rp07" gives

```
00018: /dev/hp0a:/:rw:1:1
00021: /dev/hp0g:/mnt:rw:1:3
00021: /dev/hp0h:/usr:rw:1:2
```

If the –m flag is given, then instead of printing every line in the file, *linelen* prints only the line of maximal length. For example, "linelen -m /etc/fstab.rp07" gives

```
longest line at line #2, 21 characters:
/dev/hp0g:/mnt:rw:1:3
```

AUTHOR

Jeffrey Mogul

SEE ALSO

cat(1), wc(1)

BUGS

The –m option prints the first line of maximal length; it does not indicate how many maximal-length lines there are.

Lines longer than 10000 characters will break this program.

NAME

lint − a C program verifier

SYNOPSIS

lint [−abchnpuvx] file ...

DESCRIPTION

Lint attempts to detect features of the C program *files* which are likely to be bugs, or non-portable, or wasteful. It also checks the type usage of the program more strictly than the compilers. Among the things which are currently found are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

By default, it is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to *lint*; these libraries are referred to by a conventional name, such as '−lm', in the style of *ld*(1). Arguments ending in *.ln* are also treated as library files. To create lint libraries, use the **−C** option:

lint −Cfoo files . . .

where *files* are the C sources of library *foo*. The result is a file *llib-lfoo.ln* in the correct library format suitable for linting programs using *foo*.

Any number of the options in the following list may be used. The **−D**, **−U**, and **−I** options of *cc*(1) are also recognized as separate arguments.

p Attempt to check portability to the *IBM* and *GCOS* dialects of C.

h Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.

b Report *break* statements that cannot be reached. (This is not the default because, unfortunately, most *lex* and many *yacc* outputs produce dozens of such comments.)

v Suppress complaints about unused arguments in functions.

x Report variables referred to by extern declarations, but never used.

a Report assignments of long values to int variables.

c Complain about casts which have questionable portability.

u Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running *lint* on a subset of files out of a larger program).

n Do not check compatibility against the standard library.

z Do not complain about structures that are never defined (e.g. using a structure pointer without knowing its contents.).

Exit(2) and other functions which do not return are not understood; this causes various lies.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
at appropriate points stops comments about unreachable code.

/*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.

/*NOSTRICT*/
shuts off strict type checking in the next expression.

/*ARGSUSED*/
turns on the -v option for the next function.

/*LINTLIBRARY*/
at the beginning of a file shuts off complaints about unused functions in this file.

AUTHOR

S.C. Johnson. Lint library construction implemented by Edward Wang.

FILES

/usr/lib/lint/lint[12]	programs
/usr/lib/lint/llib-lc.ln	declarations for standard functions
/usr/lib/lint/llib-lc	human readable version of above
/usr/lib/lint/llib-port.ln	declarations for portable functions
/usr/lib/lint/llib-port	human readable . . .
llib-l*.ln	library created with -C

SEE ALSO

cc(1)
S. C. Johnson, *Lint, a C Program Checker*

BUGS

There are some things you just can't get lint to shut up about.

NAME

lisp — lisp interpreter

SYNOPSIS

lisp

DESCRIPTION

Lisp is a lisp interpreter for a dialect which closely resembles MIT's MACLISP. This lisp, known as FRANZ LISP, features an I/O facility which allows the user to change the input and output syntax, add macro characters, and maintain compatibility with upper-case only lisp systems; infinite precision integer arithmetic, and an error facility which allows the user to trap system errors in many different ways. Interpreted functions may be mixed with code compiled by *liszt*(1) and both may be debugged using the "Joseph Lister" trace package. A *lisp* containing compiled and interpreted code may be dumped into a file for later use.

There are too many functions to list here; one should refer to the manuals listed below.

AUTHORS

An early version was written by Jeff Levinsky, Mike Curry, and John Breedlove. Keith Sklower wrote and is maintaining the current version, with the assistance of John Foderaro. The garbage collector was implemented by Bill Rowan.

FILES

/usr/lib/lisp/trace.l	Joseph Lister trace package
/usr/lib/lisp/toplevel.l	top level read-eval-print loop

SEE ALSO

liszt(1), lxref(1)
'FRANZ LISP Manual, Version 1' by John K. Foderaro
MACLISP Manual

BUGS

The error system is in a state of flux and not all error messages are as informative as they could be.

NAME

liszt − compile a Franz Lisp program

SYNOPSIS

liszt [−mpqruwxCQST] [−e form **] [−o** objfile **] [** name **]**

DESCRIPTION

Liszt takes a file whose names ends in '.l' and compiles the FRANZ LISP code there leaving an object program on the file whose name is that of the source with '.o' substituted for '.l'.

The following options are interpreted by *liszt.*

−e Evaluate the given form before compilation begins.

−m Compile a MACLISP file, by changing the readtable to conform to MACLISP syntax and including a macro-defined compatibility package.

−o Put the object code in the specified file, rather than the default '.o' file.

−p places profiling code at the beginning of each non-local function. If the lisp system is also created with profiling in it, this allows function calling frequency to be determined (see *prof*(1).)

−q Only print warning and error messages. Compilation statistics and notes on correct but unusual constructs will not be printed.

−r place bootstrap code at the beginning of the object file, which when the object file is executed will cause a lisp system to be invoked and the object file fasl'ed in.

−u Compile a UCI-lispfile, by changing the readtable to conform to UCI-Lisp syntax and including a macro-defined compatibility package.

−w Suppress warning diagnostics.

−x Create a lisp cross reference file with the same name as the source file but with '.x' appended. The program *lxref*(1) reads this file and creates a human readable cross reference listing.

−C put comments in the assembler output of the compiler. Useful for debugging the compiler.

-Q Print compilation statistics and warn of strange constructs. This is the default.

−S Compile the named program and leave the assembler-language output on the corresponding file suffixed '.s'. This will also prevent the assembler language file from being assembled.

−T send the assembler output to standard output.

If no source file is specified, then the compiler will run interactively. You will find yourself talking to the *lisp*(1) top-level command interpreter. You can compile a file by using the function *liszt* (an nlambda) with the same arguments as you use on the command line. For example to compile 'foo', a MACLISP file, you would use:

(liszt −m foo)

Note that *liszt* supplies the ".l" extension for you.

FILES

/usr/lib/lisp/machacks.l	MACLISP compatibility package
/usr/lib/lisp/syscall.l	macro definitions of Unix system calls
/usr/lib/lisp/ucifnc.l	UCI Lisp compatibility package

AUTHOR

John Foderaro

SEE ALSO

lisp(1), lxref(1)

NAME

ln − make links

SYNOPSIS

ln [**−s**] name1 [name2]
ln name ... directory

DESCRIPTION

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc.) may have several links to it. There are two kinds of links: hard links and symbolic links.

By default *ln* makes hard links. A hard link to a file is indistinguishable from the original directory entry; any changes to a file are effective independent of the name used to reference the file. Hard links may not span file systems and may not refer to directories.

The **−s** option causes *ln* to create symbolic links. A symbolic link contains the name of the file to which it is linked. The referenced file is used when an *open*(2) operation is performed on the link. A *stat*(2) on a symbolic link will return the linked-to file; an *lstat*(2) must be done to obtain information about the link. The *readlink*(2) call may be used to read the contents of a symbolic link. Symbolic links may span file systems and may refer to directories.

Given one or two arguments, *ln* creates a link to an existing file *name1*. If *name2* is given, the link has that name; *name2* may also be a directory in which to place the link; otherwise it is placed in the current directory. If only the directory is specified, the link will be made to the last component of *name1*.

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.

SEE ALSO

rm(1), cp(1), mv(1), link(2), readlink(2), stat(2), symlink(2)

NAME

loadavg – average load log data on a weekly basis

SYNOPSIS

loadavg

DESCRIPTION

Loadavg is a filter which reads binary load log data from standard input, and outputs one weeks worth of binary load log data that is the average

FILES

SEE ALSO

loadlog(1), exlog(1)

AUTHOR

Marc R. Hannah

BUGS

The average number of users is rounded to the nearest integer.

NAME

loadlog – log the current time, number of users, and load average

SYNOPSIS

loadlog

DESCRIPTION

Loadlog writes a binary load log entry to standard output. An entry consists of a long integer containing (time/60)*60 + nusers, where time is the result of a call to the procedure *time*(2) and nusers is the number of users currently logged in. This is followed by a short integer containing the current 15 minute load average multiplied by 100.

Loadlog is intended to be run every 15 minutes by *cron*(8) with output appended to a system log file. This can be done with the following entry in /usr/lib/crontab:

```
0,15,30,45 * * * * /usr/stanford/bin/loadlog >>/usr/adm/loadlog
```

SEE ALSO

cron(8), exlog(1), loadavg(1)

AUTHOR

Marc R. Hannah

BUGS

NAME

locate – location and owner of Pup network hosts

SYNOPSIS

locate hostname **[** hostname ... **]**

DESCRIPTION

Locate returns the "attributes" associated with the Pup Network Directory entry for the specified host(s). Usually, these attributes are "Location" and "Owner".

For example,

locate ifs

would return

Lassen (50#200#0) Location: MJ11 020, Owner: CSD

Note that the specified host name has been replaced by the "official" (prefered) name for the host.

SEE ALSO

miscserver(8)

DIAGNOSTICS

Will tell you if a host name is not found.

AUTHOR

Jeffrey Mogul

BUGS

This is a local (Stanford-only) protocol, and is fairly dumb.

NAME

lock — reserve a terminal

SYNOPSIS

lock

DESCRIPTION

Lock requests a password from the user, then prints "LOCKED" on the terminal and refuses to relinquish the terminal until the password is repeated. If the user forgets the password, he has no other recourse but to login elsewhere and kill the lock process.

AUTHOR

Kurt Shoens

BUGS

Should timeout after 15 minutes.

NAME

login − sign on

SYNOPSIS

login [username]

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 "How to Get Started" for how to dial up initially.

If *login* is invoked without an argument, it asks for a user name, and, if appropriate, a password. Echoing is turned off (if possible) during the typing of the password, so it will not appear on the written record of the session.

After a successful login, accounting files are updated and the user is informed of the existence of mail, and the message of the day is printed, as is the time he last logged in (unless he has a ".hushlogin" file in his home directory − this is mostly used to make life easier for non-human users, such as *uucp*).

Login initializes the user and group IDs and the working directory, then executes a command interpreter (usually *sh*(1)) according to specifications found in a password file. Argument 0 of the command interpreter is "−sh", or more generally the name of the command interpreter with a leading dash ("−") prepended.

Login also initializes the environment *environ*(7) with information specifying home directory, command interpreter, terminal type (if available) and user name.

If the file /etc/nologin exists *login* prints its contents on the user's terminal and exits. This is used by *shutdown*(8) to stop users logging in when the system is about to go down.

Login is recognized by *sh*(1) and *csh*(1) and executed directly (without forking).

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/spool/mail/*	mail
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/nologin	stops logins
.hushlogin	makes login quieter
/etc/securetty	lists ttys that root may log in on

SEE ALSO

init(8), getty(8), mail(1), passwd(1), passwd(5), environ(7), shutdown(8)

DIAGNOSTICS

"Login incorrect," if the name or the password is bad.
"No Shell", "cannot open password file", "no directory": consult a programming counselor.

BUGS

An undocumented option, **−r** is used by the remote login server, *rlogind*(8C) to force *login* to enter into an initial connection protocol.

NAME

look − find lines in a sorted list

SYNOPSIS

look [**−df**] string [file]

DESCRIPTION

Look consults a sorted *file* and prints all lines that begin with *string*. It uses binary search.

The options **d** and **f** affect comparisons as in *sort*(1):

d 'Dictionary' order: only letters, digits, tabs and blanks participate in comparisons.

f Fold. Upper case letters compare equal to lower case.

If no *file* is specified, */usr/dict/words* is assumed with collating sequence **−df**.

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1)

NAME

indxbib, lookbib − build inverted index for a bibliography, find references in a bibliography

SYNOPSIS

indxbib database ...
lookbib database

DESCRIPTION

Indxbib makes an inverted index to the named *databases* (or files) for use by *lookbib*(1) and *refer*(1). These files contain bibliographic references (or other kinds of information) separated by blank lines.

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a "%", followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with "%".

Indxbib is a shell script that calls /usr/lib/refer/mkey and /usr/lib/refer/inv. The first program, *mkey,* truncates words to 6 characters, and maps upper case to lower case. It also discards words shorter than 3 characters, words among the 100 most common English words, and numbers (dates) < 1900 or > 2000. These parameters can be changed; see page 4 of the *Refer* document by Mike Lesk. The second program, *inv,* creates an entry file (.ia), a posting file (.ib), and a tag file (.ic), all in the working directory.

Lookbib uses an inverted index made by *indxbib* to find sets of bibliographic references. It reads keywords typed after the ">" prompt on the terminal, and retrieves records containing all these keywords. If nothing matches, nothing is returned except another ">" prompt.

It is possible to search multiple databases, as long as they have a common index made by *indxbib.* In that case, only the first argument given to *indxbib* is specified to *lookbib.*

If *lookbib* does not find the index files (the .i[abc] files), it looks for a reference file with the same name as the argument, without the suffixes. It creates a file with a '.ig' suffix, suitable for use with *fgrep.* It then uses this fgrep file to find references. This method is simpler to use, but the .ig file is slower to use than the .i[abc] files, and does not allow the use of multiple reference files.

FILES

x.ia, *x*.ib, *x*.ic, where *x* is the first argument, or if these are not present, then *x*.ig, *x*

SEE ALSO

refer(1), addbib(1), sortbib(1), roffbib(1), lookbib(1)

BUGS

Probably all dates should be indexed, since many disciplines refer to literature written in the 1800s or earlier.

NAME

lorder — find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive (see *ar*(1)) *files*. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1).

This brash one-liner intends to build a new library from existing '.o' files.

```
ar cr library `lorder *.o | tsort`
```

The need for lorder may be vitiated by use of *ranlib*(1), which converts an ordered archive into a randomly accessed library.

FILES

*symref, *symdef
nm(1), sed(1), sort(1), join(1)

SEE ALSO

tsort(1), ld(1), ar(1), ranlib(1)

BUGS

The names of object files, in and out of libraries, must end with '.o'; nonsense results otherwise.

NAME

lorder68 – find ordering relation for an MC68000 object library

SYNOPSIS

lorder68 file ...

DESCRIPTION

The input is one or more object *files*. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld68*(1).

This brash one-liner intends to build a new library from existing '.b' files.

```
ar cr library `lorder68 *.b | tsort`
```

FILES

*symref, *symdef
nm(1), sed(1), sort(1), join(1)

SEE ALSO

tsort(1), ld68(1),

BUGS

The names of object files, in and out of libraries, must end with '.b'; nonsense results otherwise.

Doesn't handle libraries.

NAME

lower – lower the case of a filename

SYNOPSIS

lower files ...

DESCRIPTION

lower renames each of its file arguments to contain no upper case letters. Thus *lower* Mr.Bill would rename the file Mr.Bill to mr.bill. Use *lower* * to rid a directory of upper case names.

AUTHOR

V. R. Pratt

DIAGNOSTICS

An appropriate message is output if a file is not found. If renaming would overwrite an existing file the renaming is not performed and a message about the already existing file is printed.

NAME

lpq − spool queue examination program

SYNOPSIS

lpq [+[n]] [-l] [−Pprinter] [job # ...] [user ...]

DESCRIPTION

lpq examines the spooling area used by *lpd*(8) for printing files on the line printer, and reports the status of the specified jobs or all jobs associated with a user. *lpq* invoked without any arguments reports on any jobs currently in the queue. A **−P** flag may be used to specify a particular printer, otherwise the default line printer is used (or the value of the PRINTER variable in the environment). If a + argument is supplied, *lpq* displays the spool queue until it empties. Supplying a number immediately after the + sign indicates that *lpq* should sleep *n* seconds in between scans of the queue. All other arguments supplied are interpreted as user names or job numbers to filter out only those jobs of interest.

For each job submitted (i.e. invocation of *lpr*(1)) *lpq* reports the user's name, current rank in the queue, the names of files comprising the job, the job identifier (a number which may be supplied to *lprm*(1) for removing a specific job), and the total size in bytes. The **−l** option causes information about each of the files comprising the job to be printed. Normally, only as much information as will fit on one line is displayed. Job ordering is dependent on the algorithm used to scan the spooling directory and is supposed to be FIFO (First in First Out). File names comprising a job may be unavailable (when *lpr*(1) is used as a sink in a pipeline) in which case the file is indicated as "(standard input)".

If *lpq* warns that there is no daemon present (i.e. due to some malfunction), the *lpc*(8) command can be used to restart the printer daemon.

FILES

/etc/termcap	for manipulating the screen for repeated display
/etc/printcap	to determine printer characteristics
/usr/spool/*	the spooling directory, as determined from printcap
/usr/spool/*/cf*	control files specifying jobs
/usr/spool/*/lock	the lock file to obtain the currently active job

SEE ALSO

lpr(1), lprm(1), lpc(8), lpd(8)

BUGS

Due to the dynamic nature of the information in the spooling directory lpq may report unreliably. Output formatting is sensitive to the line length of the terminal; this can results in widely spaced columns.

DIAGNOSTICS

Unable to open various files. The lock file being malformed. Garbage files when there is no daemon active, but files in the spooling directory.

NAME

lpr − off line print

SYNOPSIS

lpr [**−P***printer*] [**−#***num*] [**−C** *class*] [**−J** *job*] [**−T** *title*] [**−i** [*numcols*]] [**−1234** *font*] [**−w***num*] [**−pltndgvcfrmhs**] [name ...]

DESCRIPTION

Lpr uses a spooling daemon to print the named files when facilities become available. If no names appear, the standard input is assumed. The **−P** option may be used to force output to a specific printer. Normally, the default printer is used (site dependent), or the value of the environment variable PRINTER is used.

The following single letter options are used to notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

−p Use *pr*(1) to format the files (equivalent to *print*).

−l Use a filter which allows control characters to be printed and suppresses page breaks.

−t The files are assumed to contain data from *troff*(1) (cat phototypesetter commands).

−n The files are assumed to contain data from *ditroff* (device independent troff).

−d The files are assumed to contain data from *tex*(l) (DVI format from Stanford).

−g The files are assumed to contain standard plot data as produced by the *plot*(3X) routines (see also *plot*(1G) for the filters used by the printer spooler).

−v The files are assumed to contain a raster image for devices like the Benson Varian.

−c The files are assumed to contain data produced by *cifplot*(l).

−f Use a filter which interprets the first character of each line as a standard FORTRAN carriage control character.

The remaining single letter options have the following meaning.

−r Remove the file upon completion of spooling or upon completion of printing (with the **−s** option).

−m Send mail upon completion.

−h Suppress the printing of the burst page.

−s Use symbolic links. Usually files are copied to the spool directory.

The **−C** option takes the following argument as a job classification for use on the burst page. For example,

```
lpr −C EECS foo.c
```

causes the system name (the name returned by *hostname*(1)) to be replaced on the burst page by EECS, and the file foo.c to be printed.

The **−J** option takes the following argument as the job name to print on the burst page. Normally, the first file's name is used.

The **−T** option uses the next argument as the title used by *pr*(1) instead of the file name.

To get multiple copies of output, use the **−#***num* option, where *num* is the number of copies desired of each file named. For example,

```
lpr −#3 foo.c bar.c more.c
```

would result in 3 copies of the file foo.c, followed by 3 copies of the file bar.c, etc. On the other hand,

```
cat foo.c bar.c more.c | lpr -#3
```

will give three copies of the concatenation of the files.

The **-i** option causes the output to be indented. If the next argument is numeric, it is used as the number of blanks to be printed before each line; otherwise, 8 characters are printed.

The **-w** option takes the immediately following number to be the page width for *pr*.

The **-s** option will use *symlink*(2) to link data files rather than trying to copy them so large files can be printed. This means the files should not be modified or removed until they have been printed.

The option **-1234** Specifies a font to be mounted on font position *i*. The daemon will construct a *.railmag* file referencing */usr/lib/vfont/name.size*.

FILES

/etc/passwd	personal identification
/etc/printcap	printer capabilities data base
/usr/lib/lpd*	line printer daemons
/usr/spool/*	directories used for spooling
/usr/spool/*/cf*	daemon control files
/usr/spool/*/df*	data files specified in "cf" files
/usr/spool/*/tf*	temporary copies of "cf" files

SEE ALSO

lpq(1), lprm(1), pr(1), symlink(2), printcap(5), lpc(8), lpd(8)

DIAGNOSTICS

If you try to spool too large a file, it will be truncated. *Lpr* will object to printing binary files. If a user other than root prints a file and spooling is disabled, *lpr* will print a message saying so and will not put jobs in the queue. If a connection to *lpd* on the local machine cannot be made, *lpr* will say that the daemon cannot be started. Diagnostics may be printed in the daemon's log file regarding missing spool files by *lpd*.

BUGS

Fonts for *troff* and *tex* reside on the host with the printer. It is currently not possible to use local font libraries.

NAME

lprm − remove jobs from the line printer spooling queue

SYNOPSIS

lprm [**−P***printer*] [−] [job # ...] [user ...]

DESCRIPTION

Lprm will remove a job, or jobs, from a printer's spool queue. Since the spooling directory is protected from users, using *lprm* is normally the only method by which a user may remove a job.

Lprm without any arguments will delete the currently active job if it is owned by the user who invoked *lprm*.

If the − flag is specified, *lprm* will remove all jobs which a user owns. If the super-user employs this flag, the spool queue will be emptied entirely. The owner is determined by the user's login name and host name on the machine where the *lpr* command was invoked.

Specifying a user's name, or list of user names, will cause *lprm* to attempt to remove any jobs queued belonging to that user (or users). This form of invoking *lprm* is useful only to the super-user.

A user may dequeue an individual job by specifying its job number. This number may be obtained from the *lpq*(1) program, e.g.

```
% lpq -l

1st: ken                        [job #013ucbarpa]
    (standard input)            100 bytes
% lprm 13
```

Lprm will announce the names of any files it removes and is silent if there are no jobs in the queue which match the request list.

Lprm will kill off an active daemon, if necessary, before removing any spooling files. If a daemon is killed, a new one is automatically restarted upon completion of file removals.

The **−P** option may be usd to specify the queue associated with a specific printer (otherwise the default printer, or the value of the PRINTER variable in the environment is used).

FILES

/etc/printcap	printer characteristics file
/usr/spool/*	spooling directories
/usr/spool/*/lock	lock file used to obtain the pid of the current daemon and the job number of the currently active job

SEE ALSO

lpr(1), lpq(1), lpd(8)

DIAGNOSTICS

"Permission denied" if the user tries to remove files other than his own.

BUGS

Since there are race conditions possible in the update of the lock file, the currently active job may be incorrectly identified.

NAME

ls − list contents of directory

SYNOPSIS

ls [−acdfgilqrstu1ACLFR] name ...

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, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents.

There are a large number of options:

−l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by "−>".

−g Include the group ownership of the file in a long output.

−t Sort by time modified (latest first) instead of by name.

−a List all entries; in the absence of this option, entries whose names begin with a period (.) are *not* listed.

−s Give size in kilobytes of each file.

−d If argument is a directory, list only its name; often used with **−l** to get the status of a directory.

−L If argument is a symbolic link, list the file or directory the link references rather than the link itself.

−r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

−u Use time of last access instead of last modification for sorting (with the **−t** option) and/or printing (with the **−l** option).

−c Use time of file creation for sorting or printing.

−i For each file, print the i-number in the first column of the report.

−f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **−l, −t, −s,** and **−r,** and turns on **−a;** the order is the order in which entries appear in the directory.

−F cause directories to be marked with a trailing '/', sockets with a trailing '=', symbolic links with a trailing '@', and executable files with a trailing '*'.

−R recursively list subdirectories encountered.

−1 force one entry per line output format; this is the default when output is not to a terminal.

−C force multi-column output; this is the default when output is to a terminal.

−q force printing of non-graphic characters in file names as the character '?'; this is the default when output is to a terminal.

The mode printed under the **−l** option contains 11 characters which are interpreted as follows: the first character is

d if the entry is a directory;
b if the entry is a block-type special file;

c if the entry is a character-type special file;
l if the entry is a symbolic link;
s if the entry is a socket, or
- if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, 'execute' permission is interpreted to mean permission to search the directory. The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
- if the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has the set-group-id bit set; likewise the user-execute permission character is given as **s** if the file has the set-user-id bit set.

The last character of the mode (normally 'x' or '-') is **t** if the 1000 bit of the mode is on. See *chmod*(1) for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

FILES

/etc/passwd to get user id's for 'ls -l'.
/etc/group to get group id's for 'ls -g'.

BUGS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as "ls -s" is much different than "ls -s | lpr". On the other hand, not doing this setting would make old shell scripts which used *ls* almost certain losers.

NAME

lxref − lisp cross reference program

SYNOPSIS

lxref [**−N**] xref-file ... [**−a** source-file ...]

DESCRIPTION

Lxref reads cross reference file(s) written by the lisp compiler *liszt* and prints a cross reference listing on the standard output. *Liszt* will create a cross reference file during compilation when it is given the **−x** switch. Cross reference files usually end in '.x' and consequently *lxref* will append a '.x' to the file names given if necessary. The first option to *lxref* is a decimal integer, N, which sets the *ignorelevel*. If a function is called more than *ignorelevel* times, the cross reference listing will just print the number of calls instead of listing each one of them. The default for *ignorelevel* is 50.

The **−a** option causes *lxref* to put limited cross reference information in the sources named. *lxref* will scan the source and when it comes across a definition of a function (that is a line beginning with *'(def'* it will preceed that line with a list of the functions which call this function, written as a comment preceeded by *';.. '*. All existing lines beginning with *';.. '* will be removed from the file. If the source file contains a line beginning *';.-'* then this will disable this annotation process from this point on until a *';.+'* is seen (however, lines beginning with *';.. '* will continue to be deleted). After the annoation is done, the original file *'foo.l'* is renamed to " *'#.foo.l*" and the new file with annotation is named *'foo.l'*

AUTHOR

John Foderaro

SEE ALSO

lisp(1), liszt(1)

BUGS

NAME

m4 − macro processor

SYNOPSIS

m4 [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 arguments, or if an argument is '−', the standard input is read. The processed text is written on the standard output.

Macro calls have the form

name(arg1,arg2, . . . , argn)

The '(' must immediately follow the name of the macro. If a defined macro name is not followed by a '(', it is deemed to have no arguments. Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore '_', where the first character is not a digit.

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. 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.

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 The second argument is installed 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.

undefine removes the definition of the macro named in its argument.

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 versions of *m4*.

changequote
Change quote characters to the first and second arguments. *Changequote* without arguments restores the original values (i.e., `').

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 newline.

ifelse has 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.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation); relationals; parentheses.

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 command given in the first argument. No value is returned.

maketemp fills in a string of XXXXX in its argument with the current process id.

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.

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The M4 Macro Processor*

NAME

macget – receive file from macintosh via modem7/macterminal

SYNOPSIS

macget [–rdu] [file]

DESCRIPTION

Macget receives a file from a Macintosh running MacTerminal. The File Transfer settings should specify the "Modem7" transfer method and a "MacTerminal" remote system. This program is designed for use with the 0.5 Beta and newer versions of MacTerminal, but includes a compatibility option for the older -0.15X Almost-Alpha version.

To use this program, log into the unix system using MacTerminal, start macget with the desired options, select "Send File..." from the "File" menu, and open the file you wish to send. If MacTerminal is properly configured, it will put up an indicator showing how much of the file has been transfered. Several Control-X's may be used to force macget to give up if the transfer fails.

The optional *file* parameter specifies the name to use when creating the unix files, otherwise the Mac file name is used (with spaces converted to underscores).

If none of the **–rdu** flags are specified, *macget* receives three files from the Mac: *file*.**info**, *file*.**data**, and *file*.**rsrc**. This mode is useful for storing Mac files so they can be restored later using *macput*.

The **–r** flag specifies *resource* mode. Only *file*.**rsrc** will be created, from the Mac file's resource fork.

The **–d** flag specifies *data* mode. Only *file*.**data** will be created, containing the data fork of the Mac file.

The **–u** flag requests *unix* mode, in which carriage returns are converted into unix newline characters, and the unix file *file*.**text** is created. A file saved from Mac applications as "text only" can be transfered using this option to convert it to a normal unix text file.

The **–o** flag specifies "old" (version -0.15X) MacTerminal compatibility mode. You must manually disable XON/XOFF flow control in this version to perform file transfer; this is done automatically in the newer versions.

SEE ALSO

macput(local)

BUGS

Doesn't work over flow controlled communication lines, or when using rlogin.

AUTHOR

Dave Johnson, Brown 7/31/84

NAME

macput -- send file to macintosh via modem7/macterminal

SYNOPSIS

macput file
macput [**−rdu**] file [**−t** type] [**−a** author] [**−n** name]

DESCRIPTION

Macput sends a file to a Macintosh running MacTerminal. The File Transfer settings should specify the "Modem7" transfer method and a "MacTerminal" remote system. This program is designed for use with the 0.5 Beta and newer versions of MacTerminal, but includes a compatibility option for the older -0.15X Almost-Alpha version.

To use this program, log into the unix system using MacTerminal, and run macput specifying the desired options and one file to be sent. If MacTerminal is properly configured, it will recognize that a file is arriving on the serial line and put up an indicator showing how much of the file has been sent. Several Control-X's may be used to force macput to give up if the transfer fails.

If none of the **−rdu** flags are specified, *macput* sends three files to the mac: *file*.**info**, *file*.**data**, and *file*.**rsrc**. This is useful for returning files to the mac which were stored using macget.

The **−r** flag specifies *resource* mode. Either *file*.**rsrc** or *file* will be sent to the Mac, along with a forged **.info** file and an empty **.data** file. The file sent becomes the resource fork of the Mac file.

The **−d** flag specifies *data* mode. Either *file*.**data** , *file*.**text** or *file* will be sent to the Mac, along with a forged **.info** file and an empty **.rsrc** file. The file sent becomes the data fork of the Mac file.

The **−u** flag requests *unix* mode, which is the same as *data* mode except unix newline characters are converted into carriage returns. Human-readable unix text files sent to the Mac using this option will be compatible with applications which expect "text only" files.

The **−o** flag specifies "old" (version -0.15X) MacTerminal compatibility mode. You must manually disable XON/XOFF flow control in this version to perform file transfer; this is done automatically in the newer versions.

The remaining options serve to override the default file type, author, and file name to be used on the Mac. The default type and author for *resource* mode are "APPL" and "CCOM". *data* mode defaults are "TEXT", "????", and *unix* mode defaults are "TEXT" and "MACA".

SEE ALSO

macget(local)

BUGS

Doesn't work over flow controlled communication lines, or when using rlogin.

Doesn't set the bundle bit on resource files, to incorporate any icons into the Desk Top. Use setfile to set the bundle bit.

FEATURES

Properly initializes the Creation Date.

AUTHOR

Dave Johnson, Brown 7/31/84

NAME

mail − send and receive mail

SYNOPSIS

mail [**−v**] [**−i**] [**−n**] [**−s** subject] [user ...]
mail [**−v**] [**−i**] [**−n**] **−f** [name]
mail [**−v**] [**−i**] [**−n**] **−u** user

INTRODUCTION

Mail is a intelligent mail processing system, which has a command syntax reminiscent of *ed* with lines replaced by messages.

The **−v** flag puts mail into verbose mode; the details of delivery are displayed on the users terminal. The **−i** flag causes tty interrupt signals to be ignored. This is particularly useful when using *mail* on noisy phone lines. The **−n** flag inhibits the reading of /usr/lib/Mail.rc.

Sending mail. To send a message to one or more other people, *mail* can be invoked with arguments which are the names of people to send to. You are then expected to type in your message, followed by an EOT (control−D) at the beginning of a line. A subject may be specified on the command line by using the **−s** flag. (Only the first argument after the **−s** flag is used as a subject; be careful to quote subjects containing spaces.) The section below, labeled *Replying to or originating mail,* describes some features of *mail* available to help you compose your letter.

Reading mail. In normal usage *mail* is given no arguments and checks your mail out of the post office, then prints out a one line header of each message there. The current message is initially the first message (numbered 1) and can be printed using the **print** command (which can be abbreviated **p**). You can move among the messages much as you move between lines in *ed*, with the commands '+' and '−' moving backwards and forwards, and simple numbers.

Disposing of mail. After examining a message you can **delete** (**d**) the message or **reply** (**r**) to it. Deletion causes the *mail* program to forget about the message. This is not irreversible; the message can be **undeleted** (**u**) by giving its number, or the *mail* session can be aborted by giving the **exit** (**x**) command. Deleted messages will, however, usually disappear never to be seen again.

Specifying messages. Commands such as **print** and **delete** can be given a list of message numbers as arguments to apply to a number of messages at once. Thus "delete 1 2" deletes messages 1 and 2, while "delete 1−5" deletes messages 1 through 5. The special name "*" addresses all messages, and "$" addresses the last message; thus the command **top** which prints the first few lines of a message could be used in "top *" to print the first few lines of all messages.

Replying to or originating mail. You can use the **reply** command to set up a response to a message, sending it back to the person who it was from. Text you then type in, up to an end-of-file, defines the contents of the message. While you are composing a message, *mail* treats lines beginning with the character "~" specially. For instance, typing "~m" (alone on a line) will place a copy of the current message into the response right shifting it by a tabstop. Other escapes will set up subject fields, add and delete recipients to the message and allow you to escape to an editor to revise the message or to a shell to run some commands. (These options are given in the summary below.)

Ending a mail processing session. You can end a *mail* session with the **quit** (**q**) command. Messages which have been examined go to your *mbox* file unless they have been deleted in which case they are discarded. Unexamined messages go back to the post office. The **−f** option causes *mail* to read in the contents of your *mbox* (or the specified file) for processing; when you **quit**, *mail* writes undeleted messages back to this file. The **−u** flag is a short way of doing "mail **−f** /usr/spool/mail/user".

Personal and systemwide distribution lists. It is also possible to create a personal distribution lists so that, for instance, you can send mail to "cohorts" and have it go to a group of people. Such lists can be defined by placing a line like

alias cohorts bill ozalp jkf mark kridle@ucbcory

in the file .mailrc in your home directory. The current list of such aliases can be displayed with the **alias (a)** command in *mail*. System wide distribution lists can be created by editing /usr/lib/aliases, see *aliases*(5) and *sendmail*(8); these are kept in a different syntax. In mail you send, personal aliases will be expanded in mail sent to others so that they will be able to **reply** to the recipients. System wide *aliases* are not expanded when the mail is sent, but any reply returned to the machine will have the system wide alias expanded as all mail goes through *sendmail*.

Network mail (ARPA, UUCP, Berknet) See *mailaddr(7)* for a description of network addresses.

Mail has a number of options which can be set in the *.mailrc* file to alter its behavior; thus "set askcc" enables the "askcc" feature. (These options are summarized below.)

SUMMARY

(Adapted from the 'Mail Reference Manual')

Each command is typed on a line by itself, and may take arguments following the command word. The command need not be typed in its entirety — the first command which matches the typed prefix is used. For commands which take message lists as arguments, if no message list is given, then the next message forward which satisfies the command's requirements is used. If there are no messages forward of the current message, the search proceeds backwards, and if there are no good messages at all, *mail* types "No applicable messages" and aborts the command.

— Goes to the previous message and prints it out. If given a numeric argument *n*, goes to the *n*-th previous message and prints it.

? Prints a brief summary of commands.

! Executes the UNIX shell command which follows.

Print (P) Like **print** but also prints out ignored header fields. See also **print** and **ignore.**

Reply (R) Reply to originator. Does not reply to other recipients of the original message.

Type (T) Identical to the **Print** command.

alias (a) With no arguments, prints out all currently-defined aliases. With one argument, prints out that alias. With more than one argument, creates an new or changes an on old alias.

alternates (alt) The **alternates** command is useful if you have accounts on several machines. It can be used to inform *mail* that the listed addresses are really you. When you **reply** to messages, *mail* will not send a copy of the message to any of the addresses listed on the *alternates* list. If the **alternates** command is given with no argument, the current set of alternate names is displayed.

chdir (c) Changes the user's working directory to that specified, if given. If no directory is given, then changes to the user's login directory.

copy (co) The **copy** command does the same thing that **save** does, except that it does not mark the messages it is used on for deletion when you quit.

delete (d) Takes a list of messages as argument and marks them all as deleted. Deleted messages will not be saved in *mbox*, nor will they be available for most other commands.

dp (also **dt**) Deletes the current message and prints the next message. If there is no

next message, *mail* says "at EOF."

edit (**e**) Takes a list of messages and points the text editor at each one in turn. On return from the editor, the message is read back in.

exit (**ex** or **x**) Effects an immediate return to the Shell without modifying the user's system mailbox, his *mbox* file, or his edit file in **−f**.

file (**fi**) The same as **folder**.

folders List the names of the folders in your folder directory.

folder (**fo**) The **folder** command switches to a new mail file or folder. With no arguments, it tells you which file you are currently reading. If you give it an argument, it will write out changes (such as deletions) you have made in the current file and read in the new file. Some special conventions are recognized for the name. # means the previous file, % means your system mailbox, %user means user's system mailbox, & means your ~/mbox file, and +folder means a file in your folder directory.

from (**f**) Takes a list of messages and prints their message headers.

headers (**h**) Lists the current range of headers, which is an 18 message group. If a "+" argument is given, then the next 18 message group is printed, and if a "−" argument is given, the previous 18 message group is printed.

help A synonym for ?

hold (**ho**, also **preserve**) Takes a message list and marks each message therein to be saved in the user's system mailbox instead of in *mbox*. Does not override the **delete** command.

ignore Add the list of header fields named to the *ignored list*. Header fields in the ignore list are not printed on your terminal when you print a message. This command is very handy for suppression of certain machine-generated header fields. The **Type** and **Print** commands can be used to print a message in its entirety, including ignored fields. If **ignore** is executed with no arguments, it lists the current set of ignored fields.

mail (**m**) Takes as argument login names and distribution group names and sends mail to those people.

mbox Indicate that a list of messages be sent to *mbox* in your home directory when you quit. This is the default action for messages if you do *not* have the *hold* option set.

next (**n** like **+** or CR) Goes to the next message in sequence and types it. With an argument list, types the next matching message.

preserve (**pre**) A synonym for **hold**.

print (**p**) Takes a message list and types out each message on the user's terminal.

quit (**q**) Terminates the session, saving all undeleted, unsaved messages in the user's *mbox* file in his login directory, preserving all messages marked with **hold** or **preserve** or never referenced in his system mailbox, and removing all other messages from his system mailbox. If new mail has arrived during the session, the message "You have new mail" is given. If given while editing a mailbox file with the **−f** flag, then the edit file is rewritten. A return to the Shell is effected, unless the rewrite of edit file fails, in which case the user can escape with the **exit** command.

reply (**r**) Takes a message list and sends mail to the sender and all recipients of the specified message. The default message must not be deleted.

respond A synonym for **reply**.

save (**s**) Takes a message list and a filename and appends each message in turn to the end of the file. The filename in quotes, followed by the line count and character count is echoed on the user's terminal.

set (**se**) With no arguments, prints all variable values. Otherwise, sets option. Arguments are of the form "option=value" or "option."

shell (**sh**) Invokes an interactive version of the shell.

size Takes a message list and prints out the size in characters of each message.

source (**so**) The **source** command reads *mail* commands from a file.

top Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the variable **toplines** and defaults to five.

type (**t**) A synonym for **print**.

unalias Takes a list of names defined by **alias** commands and discards the remembered groups of users. The group names no longer have any significance.

undelete (**u**) Takes a message list and marks each one as *not* being deleted.

unset Takes a list of option names and discards their remembered values; the inverse of **set**.

visual (**v**) Takes a message list and invokes the display editor on each message.

write (**w**) A synonym for **save**.

xit (**x**) A synonym for **exit**.

z *Mail* presents message headers in windowfuls as described under the **headers** command. You can move *mail*'s attention forward to the next window with the **z** command. Also, you can move to the previous window by using **z−**.

Here is a summary of the tilde escapes, which are used when composing messages to perform special functions. Tilde escapes are only recognized at the beginning of lines. The name "tilde escape" is somewhat of a misnomer since the actual escape character can be set by the option **escape**.

~!command Execute the indicated shell command, then return to the message.

~c name ... Add the given names to the list of carbon copy recipients.

~d Read the file "dead.letter" from your home directory into the message.

~e Invoke the text editor on the message collected so far. After the editing session is finished, you may continue appending text to the message.

~f messages Read the named messages into the message being sent. If no messages are specified, read in the current message.

~h Edit the message header fields by typing each one in turn and allowing the user to append text to the end or modify the field by using the current terminal erase and kill characters.

~m messages
Read the named messages into the message being sent, shifted right one tab. If no messages are specified, read the current message.

~p Print out the message collected so far, prefaced by the message header fields.

~q Abort the message being sent, copying the message to "dead.letter" in your home directory if **save** is set.

~r filename Read the named file into the message.

~s string Cause the named string to become the current subject field.

~t name ... Add the given names to the direct recipient list.

~v Invoke an alternate editor (defined by the VISUAL option) on the message collected so far. Usually, the alternate editor will be a screen editor. After you quit the editor, you may resume appending text to the end of your message.

~w filename Write the message onto the named file.

~|command Pipe the message through the command as a filter. If the command gives no output or terminates abnormally, retain the original text of the message. The command *fmt*(1) is often used as *command* to rejustify the message.

~~string Insert the string of text in the message prefaced by a single ~. If you have changed the escape character, then you should double that character in order to send it.

Options are controlled via the **set** and **unset** commands. Options may be either binary, in which case it is only significant to see whether they are set or not, or string, in which case the actual value is of interest. The binary options include the following:

append Causes messages saved in *mbox* to be appended to the end rather than prepended. (This is set in /usr/lib/Mail.rc on version 7 systems.)

ask Causes *mail* to prompt you for the subject of each message you send. If you respond with simply a newline, no subject field will be sent.

askcc Causes you to be prompted for additional carbon copy recipients at the end of each message. Responding with a newline indicates your satisfaction with the current list.

autoprint Causes the **delete** command to behave like **dp** – thus, after deleting a message, the next one will be typed automatically.

debug Setting the binary option *debug* is the same as specifying **–d** on the command line and causes *mail* to output all sorts of information useful for debugging *mail*.

dot The binary option *dot* causes *mail* to interpret a period alone on a line as the terminator of a message you are sending.

hold This option is used to hold messages in the system mailbox by default.

ignore Causes interrupt signals from your terminal to be ignored and echoed as @'s.

ignoreeof An option related to *dot* is *ignoreeof* which makes *mail* refuse to accept a control-d as the end of a message. *Ignoreeof* also applies to *mail* command mode.

metoo Usually, when a group is expanded that contains the sender, the sender is removed from the expansion. Setting this option causes the sender to be included in the group.

nosave Normally, when you abort a message with two RUBOUT, *mail* copies the partial letter to the file "dead.letter" in your home directory. Setting the binary option *nosave* prevents this.

quiet Suppresses the printing of the version when first invoked.

verbose Setting the option *verbose* is the same as using the **–v** flag on the command line. When mail runs in verbose mode, the actual delivery of messages is displayed on he users terminal.

The following options have string values:

EDITOR — Pathname of the text editor to use in the **edit** command and ~e escape. If not defined, then a default editor is used.

SHELL — Pathname of the shell to use in the ! command and the ~! escape. A default shell is used if this option is not defined.

VISUAL — Pathname of the text editor to use in the **visual** command and ~v escape.

crt — The valued option *crt* is used as a threshold to determine how long a message must be before *more* is used to read it.

escape — If defined, the first character of this option gives the character to use in the place of ~ to denote escapes.

folder — The name of the directory to use for storing folders of messages. If this name begins with a '/', *mail* considers it to be an absolute pathname; otherwise, the folder directory is found relative to your home directory.

record — If defined, gives the pathname of the file used to record all outgoing mail. If not defined, then outgoing mail is not so saved.

toplines — If defined, gives the number of lines of a message to be printed out with the **top** command; normally, the first five lines are printed.

FILES

/usr/spool/mail/*	post office
~/mbox	your old mail
~/.mailrc	file giving initial mail commands
/tmp/R#	temporary for editor escape
/usr/lib/Mail.help*	help files
/usr/lib/Mail.rc	system initialization file
Message*	temporary for editing messages

SEE ALSO

binmail(1), fmt(1), newaliases(1), aliases(5),
mailaddr(7), sendmail(8)
'The Mail Reference Manual'

BUGS

There are many flags that are not documented here. Most are not useful to the general user.
Usually, *mail* is just a link to *Mail*, which can be confusing.

AUTHOR

Kurt Shoens

NAME

mailcheck -- find out if a user has mail at a PUP host

SYNOPSIS

mailcheck [host] [user]

DESCRIPTION

Mailcheck is used to determine if a specified user has unread (**Laurel** -style) mail at a host on the Ethernet. Either the host or the user argument can be omitted, although if both are present they must be in the correct order. If the user name is omitted, the login name of the current user is assumed. If the host name is omitted, the "default" server on the network is assumed.

To make this program useful in a login file, it returns an exit status code depending on what it finds: 0 for "No new mail", 1 for "New Mail", 255 for "Error" (255 = -1 in 8 bit numbers.)

AUTHOR

Jeffrey Mogul

SEE ALSO

fetch(1)

DIAGNOSTICS

Generally self-explanatory -- they have to do with not being able to find the specified host, or not being able to get a reasonable response from it.

BUGS

The algorithm used to determine whether a single argument is a user name or a host name will break down if these name spaces overlap; the name is assumed to be a host name, but will be taken as a user name when no host of that name is on the net. Thus, you must specify the host explicitly if you want to find out about mail for a user named "IFS" or "Dover" or "MAXC", etc.

NAME

make − maintain program groups

SYNOPSIS

make [**−f** makefile] [option] ... file ...

DESCRIPTION

Make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **−f** option is present, 'makefile' and 'Makefile' are tried in order. If *makefile* is '−', the standard input is taken. More than one **−f** option may appear

Make updates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated list of targets, then a colon, then a list of prerequisite files. Text following a semicolon, and all following lines that begin with a tab, are shell commands to be executed to update the target. If a name appears on the left of more than one 'colon' line, then it depends on all of the names on the right of the colon on those lines, but only one command sequence may be specified for it. If a name appears on a line with a double colon :: then the command sequence following that line is performed only if the name is out of date with respect to the names to the right of the double colon, and is not affected by other double colon lines on which that name may appear.

Two special forms of a name are recognized. A name like *a*(*b*) means the file named *b* stored in the archive named *a*. A name like *a*((*b*)) means the file stored in archive *a* containing the entry point *b*.

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 '.c' files and a common file 'incl'.

```
pgm: a.o b.o
        cc a.o b.o −lm −o pgm
a.o: incl a.c
        cc −c a.c
b.o: incl b.c
        cc −c b.c
```

Makefile entries of the form

string1 = string2

are macro definitions. Subsequent appearances of $(*string1*) or ${*string1*} are replaced by *string2*. If *string1* is a single character, the parentheses or braces are optional.

Make infers prerequisites for files for which *makefile* gives no construction commands. For example, a '.c' file may be inferred as prerequisite for a '.o' file and be compiled to produce the '.o' file. Thus the preceding example can be done more briefly:

```
pgm: a.o b.o
        cc a.o b.o −lm −o pgm
a.o b.o: incl
```

Prerequisites are inferred according to selected suffixes listed as the 'prerequisites' for the special name '.SUFFIXES'; multiple lists accumulate; an empty list clears what came before. Order is significant; the first possible name for which both a file and a rule as described in the next paragraph exist is inferred. The default list is

.SUFFIXES: .out .o .c .e .r .f .y .l .s .p

The rule to create a file with suffix *s2* that depends on a similarly named file with suffix *s1* is specified as an entry for the 'target' *s1s2*. In such an entry, the special macro $* stands for the target name with suffix deleted, $@ for the full target name, $< for the complete list of prerequisites, and $? for the list of prerequisites that are out of date. For example, a rule for making optimized '.o' files from '.c' files is

```
.c.o: ; cc -c -O -o $@ $*.c
```

Certain macros are used by the default inference rules to communicate optional arguments to any resulting compilations. In particular, 'CFLAGS' is used for *cc*(1) options, 'FFLAGS' for *f77*(1) options, 'PFLAGS' for *pc*(1) options, and 'LFLAGS' and 'YFLAGS' for *lex* and *yacc*(1) options. In addition, the macro 'MFLAGS' is filled in with the initial command line options supplied to *make*. This simplifies maintaining a hierarchy of makefiles as one may then invoke *make* on makefiles in subdirectories and pass along useful options such as **-k**.

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the special target '.SILENT' is in *makefile*, or the first character of the command is '@'.

Commands returning nonzero status (see *intro*(1)) cause *make* to terminate unless the special target '.IGNORE' is in *makefile* or the command begins with <tab><hyphen>.

Interrupt and quit cause the target to be deleted unless the target is a directory or depends on the special name '.PRECIOUS'.

Other options:

-i Equivalent to the special entry '.IGNORE:'.

-k When a command returns nonzero status, abandon work on the current entry, but continue on branches that do not depend on the current entry.

-n Trace and print, but do not execute the commands needed to update the targets.

-t Touch, i.e. update the modified date of targets, without executing any commands.

-r Equivalent to an initial special entry '.SUFFIXES:' with no list.

-s Equivalent to the special entry '.SILENT:'.

FILES

makefile, Makefile

SEE ALSO

sh(1), touch(1), f77(1), pc(1)
S. I. Feldman *Make – A Program for Maintaining Computer Programs*

BUGS

Some commands return nonzero status inappropriately. Use **-i** to overcome the difficulty.
Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across newlines in *make*.

NAME

makedep – construct dependency lines for makefiles

SYNOPSIS

makedep [options **] [** source files **]**

DESCRIPTION

Makedep constructs a makefile-style dependency list showing which header files the object files constructed from the given source files depend upon. The dependency of the object file upon the source file is not indicated in the output; this dependency can normally be inferred by the *make* program.

Makedep handles nested includes properly, propagating dependencies of one header file upon another back to each object file whose source file includes the dependent header file.

The following options are accepted. In options that take an argument, the space between the option letter and the argument is optional.

-o file — Output file name. The default is "dependencies". The name "-" indicates standard output.

-I dir — Add *dir* to the include file search list. Multiple -I options accumulate, building the search list from left to right, with the system include directories added at the end. Directory names are interpreted relative to the directory from which *makedep* is invoked.

-U — Use the standard Unix header directories as the system search list. Equivalent to specifying -I/usr/include after all other -I options.

-V — Use the standard V-System header directories as the system search list. Equivalent to specifying the options -I/usr/sun/include -I/usr/local/include -I/usr/include after all other -I options.

-xV — Use the experimental V-System header directories as the system search list. Equivalent to specifying the options -I/usr/sun/xinclude -I/usr/sun/include -I/usr/local/include -I/usr/include after all other -I options.

-N — Use no system search list. Suppresses the warning message ordinarily printed when a header file cannot be found. This option is useful when you are not interested in dependencies on system include files.

-e ext — Object files have extension ".ext". Defaults to .b if -V or -xV is specified, .o otherwise.

-d — Turn on debug output. Useful only to the maintainers.

If the source files depend on any header files in standard system include directories, one of the options -U, -V, -xV, or -N should normally be specified. These four options are mutually exclusive. If none of these options is given, only the directories specified in -I options are included in the search list (as with the -N flag), but warning messages are still printed for any header files that cannot be found.

SEE ALSO

make(1)

DIAGNOSTICS

A warning is printed for each included file that cannot be found. Other errors are fatal; the messages should be self-explanatory.

BUGS

Pathnames that are excessively long may be silently truncated or cause crashes.

Makedep does not know that the same file can have two different names, for example "bar.h" and "foo/../bar.h". This means it will fail to detect loops in the dependency graph if the pathnames grow in this way while it is following the loop. The loop will eventually terminate due to the previous bug, and garbage output will result.

AUTHORS

Marvin Theimer and Tim Mann, Stanford.

NAME

man − find manual information by keywords; print out the manual

SYNOPSIS

man −k keyword ...
man −f file ...
man [−] [−t] [−d] [−i] [section **]** title ...

DESCRIPTION

Man is a program which gives information from the programmers manual. It can be asked for one line descriptions of commands specified by name, or for all commands whose description contains any of a set of keywords. It can also provide on-line access to the sections of the printed manual.

When given the option **−k** and a set of keywords, *man* prints out a one line synopsis of each manual sections whose listing in the table of contents contains that keyword.

When given the option **−f** and a list of file names, *man* attempts to locate manual sections related to those files, printing out the table of contents lines for those sections.

When neither **−k** nor **−f** is specified, *man* formats a specified set of manual pages. If a section specifier is given *man* looks in the that section of the manual for the given *titles*. *Section* is an Arabic section number (3 for instance). The number may followed by a single letter classifier (1g for instance) indicating a graphics program in section 1. If *section* is omitted, *man* searches all sections of the manual, giving preference to commands over subroutines in system libraries, and printing the first section it finds, if any.

If the standard output is a teletype, or if the flag **−** is given, *man* pipes its output through *cat*(1) with the option **−s** to crush out useless blank lines, *ul*(1) to create proper underlines for different terminals, and through *more*(1) to stop after each page on the screen. Hit a space to continue, a control-D to scroll 11 more lines when the output stops.

The **−t** flag causes *man* to arrange for the specified section to be *troff'ed* to a suitable raster output device; see *vtroff*(1).

The **−d** and **−i** flags cause *man* to print the specified section using *dtroff*(1) (on the Dover) or *itroff*(1) (on the Imagen), respectively.

FILES

/usr/man/man?/*
/usr/man/cat?/*

SEE ALSO

more(1), ul(1), whereis(1), catman(8)

BUGS

The manual is supposed to be reproducible either on the phototypesetter or on a typewriter. However, on a typewriter some information is necessarily lost.

NAME

merge − three-way file merge

SYNOPSIS

merge [**-p**] file1 file2 file3

DESCRIPTION

Merge incorporates all changes that lead form *file2* to *file3* into *file1*. The result goes to std. output if **-p** is present, into *file1* otherwise. *Merge* is useful for combining separate changes to an original. Suppose *file2* is the original, and both *file1* and *file3* are modifications of *file2*. Then *merge* combines both changes.

An overlap occurs if both *file1* and *file3* have changes in a common segment of lines. *Merge* prints how many overlaps occurred, and includes both alternatives in the result. The alternatives are delimited as follows:

```
<<<<<<< file1
lines in file1
=======
lines in file3
>>>>>>> file3
```

If there are overlaps, the user should edit the result and delete one of the alternatives.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.0 ; Release Date: 82/11/25 .

SEE ALSO

diff3 (1), diff (1), rcsmerge (1), co (1).

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

Mesg with argument **n** forbids messages via *write* and *talk*(1) by revoking non-user write permission on the user's terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1), talk(1)

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkstr – create an error message file by massaging C source

SYNOPSIS

mkstr [–] messagefile prefix file ...

DESCRIPTION

Mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

Mkstr will process each of the specified *files*, placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. A typical usage of *mkstr* would be

```
mkstr pistrings xx *.c
```

This command would cause all the error messages from the C source files in the current directory to be placed in the file *pistrings* and processed copies of the source for these files to be placed in files whose names are prefixed with *xx*.

To process the error messages in the source to the message file *mkstr* keys on the string 'error("' in the input stream. Each time it occurs, the C string starting at the '"' is placed in the message file followed by a null character and a new-line character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly *cat* the error message file to see its contents. The massaged copy of the input file then contains a *lseek* pointer into the file which can be used to retrieve the message, i.e.:

```
char    efilname[] = "/usr/lib/pi_strings";
int     efil = -1;

error(a1, a2, a3, a4)
{
        char buf[256];

        if (efil < 0) {
                efil = open(efilname, 0);
                if (efil < 0) {
oops:
                        perror(efilname);
                        exit(1);
                }
        }
        if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
                goto oops;
        printf(buf, a2, a3, a4);
}
```

The optional – causes the error messages to be placed at the end of the specified message file for recompiling part of a large *mkstr* ed program.

SEE ALSO

lseek(2), xstr(1)

AUTHORS

William Joy and Charles Haley

NAME

mod -- Modula-2 compiler

SYNOPSIS

mod [options] name ...

DESCRIPTION

Mod is a Modula-2 compiler. It compiles one or more Modula-2 program or implementation modules. Definition modules are not compiled. In the absence of options, it will compile all specified modules and link them together into an executable file called *a.out*.

Each program or implementation module must be in a separate file with a name ending with ".mod". Each definition module must be in a separate file called "module.def", where "module" is the name of the module. Object files ending with ".o" compiled with *mod* or some other compiler may be specified.

File name arguments ending with ".pcd" and ".s" are assumed to be pcode and assembly language files, respectively, and are translated and assembled into object files.

The following options are available:

-c Create object files but do not link them together.

-g Generate additional symbol table information for the debugger *dbx*(1).

-i Ignore the fact that there are errors in some of the modules and continue compiling the rest of them.

-mflags
Perform intermodule checking. If an out-of-date module is encountered, recompile it using the specified "flags". The flags are separated by commas or spaces, and must be quoted if spaces are used.

-n Write out what will happen when the same command is entered without the "-n" option.

-o name
Create an executable file called "name" instead of the default "a.out".

-pg Set up object files for profiling by *gprof*(1).

-r Retain pcode and assembly language files in the current directory after compilation.

-s Use standard conventions for reserved word case, cardinal data type, and strings (See Extensions, below).

-sc Use standard conventions for cardinal data type (See Extensions, below).

-sk Use standard conventions for reserved word case (See Extensions, below).

-ss Use standard conventions for string constants (See Extensions, below).

-u Convert all identifiers and reserved words to upper case (i.e., ignore the case of identifiers and reserved words on input).

-v Print out messages saying what is happening during compilation.

-C Generate runtime checks for illegal pointers, subrange and index bounds, and variant record tags.

-Ddirectory
Use the specified directory for the phases of the compiler and the location of the standard definition modules and libraries.

-L While performing intermodule checking, ignore references to modules not specified. (This is useful when checking modules to be placed in a library).

-M Perform intermodule checking, but do not recompile if inconsistencies are found.

-Nname
While performing intermodule checking, ignore references to the module "name". (This is useful when the module "name" is not a Modula-2 module.) This option may occur multiple times.

-O Perform code optimizations.

-P Stop after generating pcode in a file ending with ".pcd".

-S Stop after generating assembly language in a file ending with ".s".

LIBRARY MODULES

By default, an import of a global module will cause the compiler to look for the definition module first in the working directory and then in the standard library directory. The standard library modules are automatically linked with the program.

The default may be overridden to specify other directories of definition modules using the MODPATH environment variable. MODPATH is set to a sequence of directory names separated by colons. Those directories will be searched in the order specified to find any definition module. The corresponding object files or libraries are specified when linking. The MODPATH environment variable may be set by the user in .login or in .modpath in the working directory. If the file ".modpath" exists in the working directory, the **mod** command will use its first line as the value of the MODPATH variable.

The following modules are provided by this implementation of Modula-2. Note that system, memory, io, and bitoperations are builtin modules; definition modules for them are provided for documentation purposes only. Only strings and parameters are actually implemented in Modula-2.

system
Builtin system module. Contains types like word, address, etc., and process routines.

memory
Builtin storage module. Sets up pointers properly for runtime checks. Contains ALLOCATE and DEALLOCATE.

io
Builtin I/O module that provides formatted read and write similar to *scanf*(3) and *printf*(3).

bitoperations
Builtin bit manipulation module. Performs operations such as shift, exclusive or, etc., on integer operands.

math
Mathematical functions. Interface to the C math library.

parameters
Accesses command line parameters and environment variables.

strings
Compares, assigns, and concatenates strings.

unix
Defines some Unix system calls and C library routines.

Storage
Standard storage module, for compatibility with standard Modula-2. Contains ALLOCATE and DEALLOCATE.

DIFFERENCES AND EXTENSIONS

This implementation of Modula-2 has compiled and run Wirth's Modula-2 compiler (as modified by Cambridge University for the VAX) with only minor changes to make that compiler more portable. However, the definition of the language has been relaxed in some areas. For the most part,

these changes are upward compatible.

The following is an incomplete list of differences between this compiler and Wirth's compiler:

Reserved words and standard identifiers are recognized in any case, not just in upper case. Thus, case variations of reserved words may not be used for identifiers. This feature is disabled by the -sk option.

Cardinal and non-negative subranges that do not exceed MAXINT are considered to be subranges of integer and are compatible with integers. Subranges that exceed MAXINT are compatible with cardinal and non-negative subranges. This feature is disabled by the -sc option.

A builtin module called *io* provides formatted input and output. The *Readf* and *Writef* routines can accept any number of parameters, so long as their types correspond properly with the format string. Supported formats include: for integer and cardinal, **d**, **x**, and **o**; for real, **g** (output only), **f**, and **e**; for longreal, **G** (output only), **F**, and **E**; for char, **c**; and for string (array of char), **s** and **[]** (input only).

No import of *allocate* or *deallocate* is required to use new and dispose if the standard memory allocation routines are desired. Programs that desire checking will normall import allocate and deallocate from memory, rather than storage.

The sizes returned by *size* and *tsize* and expected by *allocate*, *deallocate* and *newprocess* are in units of bits.

The *system* module includes the type *byte*, which is analogous to *word*, as well as appropriate related constants. There is also a function cputime, which returns the accumulated program CPU time in milliseconds.

There is a standard type called *longreal* that stores a double precision real value. A standard function longfloat converts cardinals, integers, or reals to longreal.

Additional standard procedures include:

min(a,b)
: Returns the smaller of two cardinal, integer, real, or longreal values.

max(a,b)
: Returns the larger of two cardinal, integer, real, or longreal values.

assert(condition[,message])
: Aborts the program (with the optional message) if the condition is false.

number(a)
: Returns the number of elements in the specified array.

first(type)
: Returns the smallest legal value of the specified type.

last(type)
: Returns the largest legal value of the specified type.

Definition modules are not compiled.

Escape sequences may be placed in strings to specify non-printing characters. E.g., \n, \t, \r, \f, \b, \\, \', and \" mean linefeed, tab, carriage return, form feed, backspace, backslash, single quote, and double quote, respectively. In addition a \ followed by up to three octal digits specifies the the character whose ASCII code is the octal value. A single (double) quote also may be put in a string delimited with single (double) quotes by specifying two single (double) quotes. This feature is disabled by the -ss option.

The interface to Unix is through a module called *unix* rather than the *system* module. The *unixcall* procedure is handled for compatibility with the Cambridge compiler, but is not recommended.

Additional keywords are recognized in certain contexts. These keywords are prefixed by @ to avoid conflicting with valid identifiers.

Pointer attributes

Attributes may be specified between the keywords *pointer* and *to* in order to change the default assumptions of Modula-2 pointer with checking. Recognized attributes are:

@nocheck	Modula-2 pointer, no checking
@c	C/malloc pointer, no checking
@pascal	Pascal pointer, Pascal checking

Size and alignment

The size and alignment of data types may be specified preceding any type specification. The size and alignment multiples are in bits. For example,

```
type Register = @align 2 @size 4 [-8..7];
```

defines a type that occupies 4 bits aligned on a multiple of two bits. See Using Modula-2 with Unix C and Berkeley Pascal for more details.

Exports Exports from a definition module are assumed qualified whether the export statement says qualified or not. Unqualified exports are permitted if the @unqualified keyword is used. Multiple export statements are permitted, but they must occur next to each other.

External variables and procedures

A procedure or variable may be accessed by C and Pascal routines using its unqualified name if the @external attribute occurs between the keyword procedure and the name of the procedure or precedes the variable declaration, respectively. See Using Modula-2 with Unix C and Berkeley Pascal for more details.

Uncounted open arrays

Open array parameters appear as two parameters, the address of the array and the number of element, to non-Modula-2 programs. If necessary, the count may be omitted by placing the attribute @nocount between the keywords *array* and *of* in the open array declaration. See Using Modula-2 with Unix C and Berkeley Pascal for more details.

FILES

file.mod	Program or implementation module
file.def	Definition module
file.pcd	Pcode (-P or -r)
file.s	Assembly code (-S or -r)
/usr/local/lib/mod/mod2.0	Modula-2 compiler front-end
/usr/local/lib/mod/mod2.1	Modula-2 compiler back-end
/usr/local/lib/mod/mod2.2	Intermodule checker
/usr/local/lib/mod/*.def	Standard definition modules
/usr/local/lib/mod/modlib	Default library
/tmp/modNNNNNN.pcd	Temporary Pcode file
/tmp/modNNNNNN.s	Temporary assembly code file

SEE ALSO

N. Wirth, *Programming in Modula-2*, Springer-Verlag, New York, 1982.

DIAGNOSTICS

All error messages suppress subsequent compilation phases. Error messages ending with a question mark are internal errors, and probably represent compiler bugs. When pointer checking is enabled in a running Modula-2 program, segmentation faults may be generated by the pointer validation test. These are intentional and should be considered as invalid pointer messages. The compiler runs with runtime checks enabled, and may produce core dumps. Report problems to the author.

AUTHOR

Michael L. Powell
Digital Equipment Corporation
Western Research Laboratory
4410 El Camino Real
Los Altos, CA 94022
Mail: powell@decwrl.csnet or {decvax,ucbvax}!decwrl!powell

LIMITATIONS

This is an experimental compiler, and thus no warranties are expressed or implied about its conformance to the definition of the Modula-2 language or about its proper functioning. We will endeavor to report and fix bugs, but users should be aware that this compiler is not a supported product.

NAME

more, page — file perusal filter for crt viewing

SYNOPSIS

more [**−cdflsu**] [*−n*] [*+linenumber*] [*+/pattern*] [name ...]

page *more options*

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a soft-copy terminal. It normally pauses after each screenful, printing --More-- at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. Other possibilities are enumerated later.

The command line options are:

−*n* An integer which is the size (in lines) of the window which *more* will use instead of the default.

−c *More* will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.

−d *More* will prompt the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

−f This causes *more* to count logical, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen positions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.

−l Do not treat ^L (form feed) specially. If this option is not given, *more* will pause after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

−s Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

−u Normally, *more* will handle underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* will output appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The *−u* option suppresses this processing.

+*linenumber*
Start up at *linenumber*.

+/*pattern*
Start up two lines before the line containing the regular expression *pattern*.

If the program is invoked as *page*, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and $k - 1$ rather than $k - 2$ lines are printed in each screenful, where k is the number of lines the terminal can display.

More looks in the file */etc/termcap* to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the *-c* mode of operation, the *csh* command *setenv MORE -c* or the *sh* command sequence *MORE='-c' ; export MORE* would cause all invocations of *more*, including invocations by programs such as *man* and *msgs*, to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.cshrc* or *.profile* file.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i<space>
: display *i* more lines, (or another screenful if no argument is given)

^D
: display 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d
: same as ^D (control-D)

*i*z
: same as typing a space except that *i*, if present, becomes the new window size.

*i*s
: skip *i* lines and print a screenful of lines

*i*f
: skip *i* screenfuls and print a screenful of lines

q or Q
: Exit from *more*.

=
: Display the current line number.

v
: Start up the editor *vi* at the current line.

h
: Help command; give a description of all the *more* commands.

i/expr
: search for the *i*-th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

*i*n
: search for the *i*-th occurrence of the last regular expression entered.

'
: (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
: invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n
: skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense)

i:p
: skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f
: display the current file name and line number.

:q or :Q
exit from *more* (same as q or Q).

. (dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\). *More* will stop sending output, and will display the usual --More-- 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.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then *more* acts just like *cat,* except that a header is printed before each file (if there is more than one).

A sample usage of *more* in previewing *nroff* output would be

```
nroff -ms +2 doc.n | more -s
```

AUTHOR

Eric Shienbrood, minor revisions by John Foderaro and Geoffrey Peck

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

SEE ALSO

csh(1), man(1), msgs(1), script(1), sh(1), environ(7)

NAME

msgs − system messages and junk mail program

SYNOPSIS

msgs [**−fhlpq**] [number] [−number]

DESCRIPTION

Msgs is used to read system messages. These messages are sent by mailing to the login 'msgs' and should be short pieces of information which are suitable to be read once by most users of the system.

Msgs is normally invoked each time you login, by placing it in the file *.login (.profile* if you use */bin/sh)*. It will then prompt you with the source and subject of each new message. If there is no subject line, the first few non-blank lines of the message will be displayed. If there is more to the message, you will be told how long it is and asked whether you wish to see the rest of the message. The possible responses are:

y type the rest of the message

RETURN
synonym for y.

n skip this message and go on to the next message.

− redisplay the last message.

q drops you out of *msgs;* the next time you run the program it will pick up where you left off.

s append the current message to the file "Messages" in the current directory; 's−' will save the previously displayed message. A 's' or 's−' may be followed by a space and a filename to receive the message replacing the default "Messages".

m or 'm−' causes a copy of the specified message to be placed in a temporary mailbox and *mail*(1) to be invoked on that mailbox. Both 'm' and 's' accept a numeric argument in place of the '−'.

Msgs keeps track of the next message you will see by a number in the file *.msgsrc* in your home directory. In the directory */usr/msgs* it keeps a set of files whose names are the (sequential) numbers of the messages they represent. The file */usr/msgs/bounds* shows the low and high number of the messages in the directory so that *msgs* can quickly determine if there are no messages for you. If the contents of *bounds* is incorrect it can be fixed by removing it; *msgs* will make a new *bounds* file the next time it is run.

Options to msgs include:

−f which causes it not to say "No new messages.". This is useful in your *.login* file since this is often the case here.

−q Queries whether there are messages, printing "There are new messages." if there are. The command "msgs −q" is often used in login scripts.

−h causes *msgs* to print the first part of messages only.

−l option causes only locally originated messages to be reported.

num A message number can be given on the command line, causing *msgs* to start at the specified message rather than at the next message indicated by your *.msgsrc* file. Thus

msgs −h 1

prints the first part of all messages.

−number
will cause *msgs* to start *number* messages back from the one indicated by your *.msgsrc*

file, useful for reviews of recent messages.

-p causes long messages to be piped through *more*(1).

Within *msgs* you can also go to any specific message by typing its number when *msgs* requests input as to what to do.

FILES

/usr/msgs/*	database
~/.msgsrc	number of next message to be presented

AUTHORS

William Joy
David Wasley

SEE ALSO

mail(1), more(1)

BUGS

NAME

mt − magnetic tape manipulating program

SYNOPSIS

mt [**−f** *tapename*] *command* [*count*]

DESCRIPTION

Mt is used to give commands to a magnetic tape drive. If a tape name is not specified, the environment variable TAPE is used; if TAPE does not exist, *mt* uses the device */dev/rmt12*. Note that *tapename* must reference a raw (not block) tape device. By default *mt* performs the requested operation once. Operations may be performed multiple times by specifying *count*.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

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 Back space *count* files.

bsr Back space *count* records.

rewind Rewind the tape (*Count* is ignored.)

offline, rewoffl
Rewind the tape and place the tape unit off-line (*Count* is ignored.)

status Print status information about the tape unit.

Mt returns a 0 exit status when the operation(s) were successful, 1 if the command was unrecognized, and 2 if an operation failed.

FILES

/dev/rmt* Raw magnetic tape interface

SEE ALSO

mtio(4), dd(1), ioctl(2), environ(7)

NAME

mv − move or rename files

SYNOPSIS

mv [**−i**] [**−f**] [**−**] file1 file2

mv [**−i**] [**−f**] [**−**] file ... directory

DESCRIPTION

Mv moves (changes the name of) *file1* to *file2*.

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, *mv* prints the mode (see *chmod*(2)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place; if not, *mv* exits.

In the second form, one or more *files* (plain files or directories) are moved to the *directory* with their original file-names.

Mv refuses to move a file onto itself.

Options:

−i stands for interactive mode. Whenever a move is to supercede an existing file, the user is prompted by the name of the file followed by a question mark. If he answers with a line starting with 'y', the move continues. Any other reply prevents the move from occurring.

−f stands for force. This option overrides any mode restrictions or the −i switch.

− means interpret all the following arguments to *mv* as file names. This allows file names starting with minus.

SEE ALSO

cp(1), ln(1)

BUGS

If *file1* and *file2* lie on different file systems, *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

NAME

net -- print IP net names and addresses

SYNOPSIS

net netname | net-number ...

DESCRIPTION

Net invoked with a net name or number prints the name(s) of that net and the internet network number of the net.

AUTHOR

Jeff Mogul

BUGS

This inherits lots of bugs from the library routines that look up net names.

The format of a network number is stupid.

NAME

netalias – keeping track of remote user names and passwords

SYNOPSIS

netalias [hostname **]**

DESCRIPTION

Netalias is used to specify the username and password that you normally use on a remote network host. Several programs that communicate with other computers over networks use this information so that you do not have to type it every time you run such a program. (Some of these programs will optionally store your user name and password, but you can also use *netalias* to do so.) The information is encrypted and stored in a file under your home directory with a name consisting of a dot followed by the host name in lower case. The file is "secret" in that is read protected against everyone but the owner, and normally is not printed in directory listings since it begins with a dot.

USAGE

The program carries on a dialog with the user. If you do not specify a *hostname* argument when you run *netalias*, you will be asked for one.

If you do not specify "yes" to the question:

May I encrypt and store this password ? (yes or no):

then the file is still written, but instead of your password a special code is written indicating that you should never be asked this question again for this host. You may change passwords by running *netalias* again, or by removing the file before running a network program again.

FILES

$HOME/.<hostname>

SEE ALSO

pupftp(1), fetch(1)

AUTHOR

Bill Nowicki

BUGS

****WARNING**** Do not be fooled into thinking that just because the password is "encrypted" that it will be safe. A two way encryption scheme is used instead of the one way scheme used for Unix passwords, since the original text must be easily obtainable from the encrytped text. Unfortunately this means that anyone who learns the key can decode everyone's password.

The routines do not know about aliases, so if you use two different names for the same host, they are treated as different hosts.

NAME

netsend – send a short message to one or more users on the Ethernet

SYNOPSIS

netsend hostname username

DESCRIPTION

Netsend is used to send a message of up to 500 characters to either a specific user, or all logged in users, at a given host on the Ethernet. Input is taken from the standard input, until end-of-file.

The message is delivered with the heading

Message from yourname at hostname at time ...

Messages to all users at a host can be sent by specifying '*' as a destination username; unfortunately, you must type "*" to get the '*' past the shell. Broadcast messages are delivered with the heading

Broadcast message from yourname at hostname at time ...

You may deny or grant permission to send messages to your terminal with the *mesg(1)* command. Certain commands, in particular *nroff* and *pr*(1) disallow messages in order to prevent messy output.

AUTHOR

Jeffrey Mogul

SEE ALSO

mesg(1), who(1), mail(1), write(1)

DIAGNOSTICS

Many are possible; most are self-evident. A remote host may refuse a message because the user is not logged in, because the user has denied permission to send messages, or because the remote host is being nasty. A remote host which does not understand the non-standard protocol involved will not respond at all.

If you type too much (i.e., more than 500 characters), you will be told that you have gone too far, that the last line you typed was ignored, and the message so far will be sent. You can send more by using the command all over again.

BUGS

Many.

This is a non-standard Pup protocol. Only a few machines at

Stanford are running the server process that recognizes the request.

The 500 character limit is imposed because the protocol puts everything in one Pup packet. Additionally, the entire message is delivered in one chunk; for conversations, this can be annoying. (For conversations on your local host, use *write(1)*.)

The argument syntax is awkward (the '*' character is eaten by the shell unless escaped), and inflexible. Ideally, one would like to be able to do such things as send to all users on all hosts, to a given user on whatever host, and to have the program understand free format argument syntax and defaults so that everything can be expressed naturally.

One *can* send a message to all hosts (either to a specific user or to all the users) by specifying '0' as the host name; this is a bit of a hack, since it can result in multiple deliveries to the same machine (actually, even a non-broadcast message can get delivered more than once, but the broadcast improves the chances for an error markedly.)

NAME

netstat – show network status

SYNOPSIS

netstat [**–Aahimnrs**] [**–p** *protocol*] [**–a**] [*interval*] [*system*] [*core*]

DESCRIPTION

The *netstat* command symbolically displays the contents of various network-related data structures. The options have the following meaning:

–A show the address of any associated protocol control blocks; used for debugging

–a show the state of all sockets; normally sockets used by server processes are not shown

–h show the state of the IMP host table

–i show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown)

–m show statistics recorded by the memory management routines (the network manages a "private share" of memory)

–n show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically)

–p *proto*
show the state of sockets utilizing protocol *proto*; the protocol is specified symbolically, and may be any protocol listed in the file */etc/protocols*.

–s show per-protocol statistics

–r show the routing tables

–rs show routing table statistics

The arguments, *system* and *core* allow substitutes for the defaults "/vmunix" and "/dev/kmem".

If an *interval* is specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces, pausing *interval* seconds before refreshing the screen.

There are a number of display formats, depending on the information presented. The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and, optionally, 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. When known the host and network addresses are displayed symbolically according to the data bases */etc/hosts* and */etc/networks*, respectively. If a symbolic name for an address is unknown, or if the **–n** option is specified, the address is printed in the Internet "dot format"; refer to *inet*(3N) for more information regarding this format. Unspecified, or "wildcard", addresses and ports appear as "*".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network address (currently Internet specific) of the interface and the maximum transmission unit ("mtu") are also displayed.

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 state of the route ("U" if "up"), and whether the route is to a gateway ("G"). Direct routes are created for each interface attached to the local host. 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 then discard it. 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 *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column summarizing information for all interfaces, and a column for the interface with the most traffic since the system was last rebooted. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

SEE ALSO

iostat(1), vmstat(1), hosts(5), networks(5), protocols(5), services(5), trpt(8C)

BUGS

The notion of errors is ill-defined. Collisions mean something else for the IMP.

NAME

netupd – update a directory from one on another system

SYNOPSIS

netupd [-abdfilnru **]** remote-host remote-dir **[** local-dir **]**

DESCRIPTION

Netupd is used to maintain consistency between directories on several Unix systems, connected over a Pup-based internetwork. It adds to a directory on the local host files from a directory on the remote host which are newer than the local file of the same name. Files which exist only on the remote host are copied to the local host.

The mode bits of the files are copied from the remote system; however, the ownership reflects the user running the *netupd* program. If any action is taken (or attempted unsuccessfully), it is logged in a file contained in the local directory.

The *remote-host* argument is the name of the remote system, which should be a Unix system running the FTP server. (Non-Unix systems, can be used, but the Unix mode bits will not be copied. Instead, they will reflect the mode of the local version of the file if it exists, or the prevailing *umask* otherwise.)

The *remote-dir* argument must be the full pathname (in a form that makes sense at the remote host) of the directory on the remote host. (When communicating with systems that allow "<" and ">" in directory names, you will have to quote this argument to protect it from the shell.)

If the *local-dir* argument is given, the program does an implicit *cd local-dir* before doing anything.

The flag arguments must be given as one argument; the possible arguments are:

–a	"Ask" the user if the update should take place, for each file that would normally be updated. To update files whose name is the null string, or a single space, this option must be used.
–b	Turns on BSP debugging information; usually not very useful.
–d	Turns on FTP and other higher-level debugging information. Probably not useful.
–f	Interpret the *remote-dir* argument as the full pathname of a single file, and update just that file.
–i	"Indicate" what would be done, but don't actually perform any updates.
–l	Insure that files that already exist with more than one "Link" are not updated (this is a temporary kludge in place of a solution that would preserve links.) If this flag is not given, broken links are noted in the log file.
–n	Sets the file's access-time and modify-time to "Now" instead of the time the file was written on the remote system.
–r	"Recursively" update subdirectories when encountered. Unless the remote system uses Unix-like pathnames (i.e., the "foo" subdirectory of "/bar" is "/bar/foo"), this may not work very well.
–u	Allows you to specify a username and password for use on the remote system; these will be prompted for interactively. Otherwise, the program uses your default username and password for the remote system; this can be set using the "login" command in *ftp*(1).

FILES

.netupdlog	in the directory/directories updated, contains a log of activity by the *netupd* program. This file will not be updated by this program
.hostname	on your login directory, used to determine the default username and password.

SEE ALSO

ftp(1), netalias(8)

DIAGNOSTICS

Should be obvious.

AUTHOR

Jeffrey Mogul, based on user FTP by Bill Nowicki

BUGS

Links are not preserved.

No interlocking is done on the .netupdlog file.

If the remote server is on a Unix system but is out-of-date (version 1.21 or earlier), this program won't work right. It will warn you. Complain to the system manager at the remote site.

NAME

newaliases – rebuild the data base for the mail aliases file

SYNOPSIS

newaliases

DESCRIPTION

Newaliases rebuilds the random access data base for the mail aliases file */usr/lib/aliases*. It must be run each time */usr/lib/aliases* is changed in order for the change to take effect.

SEE ALSO

aliases(5), sendmail(8)

BUGS

NAME

next – show the next message

SYNOPSIS

next [+folder] [–switches for *l*] [–help]

DESCRIPTION

Next performs a *show* on the next message in the specified (or current) folder. Like *show*, it passes any switches on to the program *l*, which is called to list the message. This command is exactly equivalent to "show next".

FILES

$HOME/mh_profile The user profile

PROFILE COMPONENTS

Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder

CONTEXT

If a folder is specified, it will become the current folder, and the message that is shown (i.e., the next message in sequence) will become the current message.

NAME

nice, nohup − run a command at low priority (*sh* only)

SYNOPSIS

nice [−*number*] command [arguments]

nohup command [arguments]

DESCRIPTION

Nice executes *command* with low scheduling priority. If the *number* argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 20. The default *number* is 10.

The super-user may run commands with priority higher than normal by using a negative priority, e.g. '−−10'.

Nohup executes *command* immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. *Nohup* should be invoked from the shell with '&' in order to prevent it from responding to interrupts by or stealing the input from the next person who logs in on the same terminal. The syntax of nice is also different.

FILES

nohup.out standard output and standard error file under *nohup*

SEE ALSO

csh(1), setpriority(2), renice(8)

DIAGNOSTICS

Nice returns the exit status of the subject command.

BUGS

Nice and *nohup* are particular to *sh*(1). If you use *csh*(1), then commands executed with "&" are automatically immune to hangup signals while in the background. There is a builtin command *nohup* which provides immunity from terminate, but it does not redirect output to *nohup.out.*

Nice is built into *csh*(1) with a slightly different syntax than described here. The form "nice +10" nices to positive nice, and "nice −10" can be used by the super-user to give a process more of the processor.

NAME

nm − print name list

SYNOPSIS

nm [**−gnopru**] [file ...]

DESCRIPTION

Nm prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in "a.out" are listed.

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), **B** (bss segment symbol), **C** (common symbol), **f** file name, or **−** for sdb symbol table entries (see **−a** below). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

−g Print only global (external) symbols.

−n Sort numerically rather than alphabetically.

−o Prepend file or archive element name to each output line rather than only once.

−p Don't sort; print in symbol-table order.

−r Sort in reverse order.

−u Print only undefined symbols.

SEE ALSO

ar(1), ar(5), a.out(5), stab(5)

NAME

nm68 – print name list of MC68000 object files

SYNOPSIS

nm68 [−cgnfopruh] [file ... **]**

DESCRIPTION

Nm68 prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced.

Each symbol name is preceded by its value in hex and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

- **−g** Print only global (external) symbols.
- **−n** Sort numerically rather than alphabetically.
- **−o** Prepend file or archive element name to each output line rather than only once.
- **−p** Don't sort; print in symbol-table order.
- **−r** Sort in reverse order.
- **−d** Print only defined symbols.
- **−u** Print only undefined symbols.
- **−h** Print values in hex rather than in octal. (**−x** is a synonym for **−h**
- **−c** Print only C-style symbols (those beginning with '~' or '_'.)
- **−f** Give the name of each object file as it is processed. This implies a **−p** flag. (If the **−p** flag is not present, nm68 sorts symbols without regard to which object file they came from.)

SEE ALSO

ar(1), ar(5), b.out(5)

NAME

nroff − text formatting

SYNOPSIS

nroff [option] ... [file] ...

DESCRIPTION

Nroff formats text in the named *files* for typewriter-like devices. See also *troff(1)*. The full capabilities of *nroff* are described in the *Nroff/Troff User's Manual.*

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (−) is taken to be a file name corresponding to the standard input.

The options, which may appear in any order so long as they appear *before* the files, are:

−o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N*−*M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end.

−n*N* Number first generated page *N*.

−s*N* Stop every *N* pages. *Nroff* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline.

−m*name* Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

−r*aN* Set register *a* (one-character) to *N*.

−i Read standard input after the input files are exhausted.

−q Invoke the simultaneous input-output mode of the **rd** request.

−T*name* Prepare output for specified terminal. Known *names* are **37** for the (default) Teletype Corporation Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300S** for the DASI-300S, **300** for the DASI-300, and **450** for the DASI-450 (Diablo Hyterm).

−e Produce equally-spaced words in adjusted lines, using full terminal resolution.

−h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

FILES

/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*

SEE ALSO

J. F. Ossanna, ***Nroff/Troff user's manual***
B. W. Kernighan, ***A TROFF Tutorial***
troff(1), eqn(1), tbl(1), ms(7), me(7), man(7), col(1)

NAME

o68 -- .s -> .s optimizer component of cc68

SYNOPSIS

o68

DESCRIPTION

O68 is a 68000 assembly language optimizer. It takes its input from stdin and sends its output to stdout.

FILES

/usr/sun/c68/o0.c /usr/sun/c68/o1.c /usr/bin/o68

SEE ALSO

cc68 (1)

NAME

od − octal, decimal, hex, ascii dump

SYNOPSIS

od [−format] [file] [[+]offset[.][**b**] [label]]

DESCRIPTION

Od displays *file*, or it's standard input, in one or more dump formats as selected by the first argument. If the first argument is missing, **−o** is the default. Dumping continues until end-of-file.

The meanings of the format argument characters are:

a Interpret bytes as characters and display them with their ACSII names. If the **p** character is given also, then bytes with even parity are underlined. The **P** character causes bytes with odd parity to be underlined. Otherwise the parity bit is ignored.

b Interpret bytes as unsigned octal.

c Interpret bytes as ASCII characters. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, formfeed=\f, newline=\n, return=\r, tab=\t; others appear as 3-digit octal numbers. Bytes with the parity bit set are displayed in octal.

d Interpret (short) words as unsigned decimal.

f Interpret long words as floating point.

h Interpret (short) words as unsigned hexadecimal.

i Interpret (short) words as signed decimal.

l Interpret long words as signed decimal.

o Interpret (short) words as unsigned octal.

s[n] Look for strings of ascii graphic characters, terminated with a null byte. *N* specifies the minimum length string to be recognized. By default, the minimum length is 3 characters.

v Show all data. By default, display lines that are identical to the last line shown are not output, but are indicated with an "*" in column 1.

w[n] Specifies the number of input bytes to be interpreted and displayed on each output line. If **w** is not specified, 16 bytes are read for each display line. If *n* is not specified, it defaults to 32.

x Interpret (short) words as hexadecimal.

An upper case format character implies the long or double precision form of the object.

The *offset* argument specifies the byte offset into the file where dumping is to commence. By default this argument is interpreted in octal. A different radix can be specified; If "." is appended to the argument, then *offset* is interpreted in decimal. If *offset* begins with "**x**" or "**0x**", it is interpreted in hexadecimal. If "**b**" ("**B**") is appended, the offset is interpreted as a block count, where a block is 512 (1024) bytes. If the *file* argument is omitted, an *offset* argument must be preceded by "**+**".

The radix of the displayed address will be the same as the radix of the *offset*, if specified; otherwise it will be octal.

Label will be interpreted as a pseudo-address for the first byte displayed. It will be shown in "()" following the file offset. It is intended to be used with core images to indicate the real memory address. The syntax for *label* is identical to that for *offset*.

SEE ALSO

adb(1)

BUGS

A file name argument can't start with "+". A hexadecimal offset can't be a block count. Only one file name argument can be given.

It is an historical botch to require specification of object, radix, and sign representation in a single character argument.

NAME

p2m2 – Pascal to Modula-2 conversion tool

SYNOPSIS

p2m2 [-h] [-s] name

DESCRIPTION

P2m2 is a tool for assisting with the conversion of programs from Berkeley Pascal to Modula-2. It does not perform a complete translation, since there are some language features in Pascal that either do not exist in Modula-2 (e.g., **goto** statements) or are too difficult to convert (e.g., write statements). It also does not necessarily produce a "good" Modula-2 program, since it does not restructure a program into modules.

In spite of these disclaimers, it is possible to use *p2m2* to rapidly convert a substantial amount of Pascal software to Modula-2.

Files are converted one at a time. A file name must end with ".p" for a Pascal program or separate compilation unit, or with ".h" for a set of definitions or external specifications for a separate compilation unit. A file called "name.p" will produce a program or implementation module called "name" in a file called "name.mod". A program module will be generated if the file contains a Pascal program. A file called "name.h" will produce a definition module called name in a file called "name.def".

The **–h** flag is specified when a separate compilation unit is converted. The option causes *p2m2* to scan the corresponding ".h" file for procedure parameter definitions and insert those in the generated ".mod" file. This flag is useful because Berkeley Pascal prohibits parameters from appearing in the procedure definition if there is an external declaration of the procedure.

The **–s** flag outputs reserved words in upper case, in accordance with the Modula-2 report. By default, *p2m2* generates reserved words in lower case.

METHOD

As *p2m2* parses the Pascal program, it copies white space (comments, spaces, new lines) to the output. It outputs tokens and identifiers, rearranging them as necessary. The result is a program that is formatted approximately the same as the original.

The names of procedures, functions, variables, types, and constants defined in ".h" files are exported from the definition module. The names are exported unqualified, to simulate the global naming that takes place in Berkeley Pascal. (Although the Modula-2 report requires global exports to be qualified, the DEC Modula-2 compiler permits unqualified exports.) An include directive is changed to a comment, but causes the named module to be imported.

UNHANDLED DIFFERENCES

The following is a partial list of differences between Pascal and Modula-2 that are not handled:

Forward declarations are not necessary, since Modula-2 allows procedures to be defined after they are used.

Goto statements and label declarations are not supported. Many goto statements may be avoided by using the **loop** and **exit** or **return** statements.

I/O and files are different, including write statements. See *mod*(l) and the standard module *io.def* for details.

Procedure parameters are supported differently. Since Modula-2 supports procedure variables (which may be passed as parameters), the syntax for formal procedure parameters is similar to other formal parameters.

Function return values are done through the return statement, not by assignment to the function name. Assignments to functions are marked with the comment (*!return!*) to allow easy editting in most cases.

FILES

file.p	Pascal main program or separate compilation unit
file.h	Pascal header file
file.mod	Program or implementation module
file.def	Definition module
p2m2.temp	Output of first pass of p2m2

SEE ALSO

N. Wirth, *Programming in Modula-2*, Springer-Verlag, New York, 1982.

DIAGNOSTICS

Error messages are written to standard output. In addition, comments are inserted in the generated modules to mark places where the translation failed. Such comments are of the form (*! ... !*).

AUTHORS

Benjamin C. Pierce
Michael L. Powell
Digital Equipment Corporation
Western Research Laboratory
4410 El Camino Real
Los Altos, CA 94022
Mail: powell@decwrl.csnet or {decvax,ucbvax}!decwrl!powell

LIMITATIONS

This is an experimental tool, and thus no warranties are expressed or implied about its conformance to the definition of the Modula-2 language or about its proper functioning.

NAME

pagesize — print system page size

SYNOPSIS

pagesize

DESCRIPTION

Pagesize prints the size of a page of memory in bytes, as returned by *getpagesize*(2). This program is useful in constructing portable shell scripts.

SEE ALSO

getpagesize(2)

NAME

passwd — change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This command changes (or installs) a password associated with the user *name* (your own name by default).

The program prompts for the old password and then for the new one. The caller must supply both. The new password must be typed 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 may change a password; the owner must prove he knows the old password.

FILES

/etc/passwd

SEE ALSO

login(1), passwd(5), crypt(3)
Robert Morris and Ken Thompson, *UNIX password security*

BUGS

The password file information should be kept in a different data structure allowing indexed access; *dbm*(3X) would probably be suitable.

NAME

pc − Pascal compiler

SYNOPSIS

pc [option] [**−i** name ...] name ...

DESCRIPTION

Pc is a Pascal compiler. If given an argument file ending with **.p**, it will compile the file and load it into an executable file called, by default, *a.out*.

A program may be separated into more than one **.p** file. *Pc* will compile a number of argument **.p** files into object files (with the extension **.o** in place of **.p**). Object files may then be loaded into an executable *a.out* file. Exactly one object file must supply a **program** statement to successfully create an executable a.out file. The rest of the files must consist only of declarations which logically nest within the program. References to objects shared between separately compiled files are allowed if the objects are declared in **included** header files, whose names must end with **.h**. Header files may only be included at the outermost level, and thus declare only globally available objects. To allow **functions** and **procedures** to be declared, an **external** directive has been added, whose use is similar to the **forward** directive but restricted to appear only in **.h** files. **Function** and **procedure** bodies may not appear in **.h** files. A binding phase of the compiler checks that declarations are used consistently, to enforce the type checking rules of Pascal.

Object files created by other language processors may be loaded together with object files created by *pc*. The **functions** and **procedures** they define must have been declared in **.h** files included by all the **.p** files which call those routines. Calling conventions are as in C, with **var** parameters passed by address.

See the Berkeley Pascal User's Manual for details.

The following options have the same meaning as in *cc*(1) and *f77*(1). See *ld*(1) for load-time options.

−c Suppress loading and produce '.o' file(s) from source file(s).

−g Have the compiler produce additional symbol table information for *dbx*(1).

−w Suppress warning messages.

−p Prepare object files for profiling, see *prof*(1).

−O Invoke an object-code improver.

−S Compile the named program, and leave the assembler-language output on the corresponding file suffixed '.s'. (No '.o' is created.).

−o output
Name the final output file *output* instead of *a.out*.

The following options are peculiar to *pc*.

−C Compile code to perform runtime checks, verify **assert** calls, and initialize all variables to zero as in *pi*.

−b Block buffer the file *output*.

−i Produce a listing for the specified procedures, functions and **include** files.

−l Make a program listing during translation.

−s Accept standard Pascal only; non-standard constructs cause warning diagnostics.

−t directory
Use the given *directory* for compiler temporary files.

−z Allow execution profiling with *pxp* by generating statement counters, and arranging for

the creation of the profile data file *pmon.out* when the resulting object is executed.

Other arguments are taken to be loader option arguments, perhaps libraries of *pc* compatible routines. Certain flags can also be controlled in comments within the program as described in the *Berkeley Pascal User's Manual.*

FILES

file.p	pascal source files
/usr/lib/pc0	compiler
/lib/f1	code generator
/usr/lib/pc2	runtime integrator (inline expander)
/lib/c2	peephole optimizer
/usr/lib/pc3	separate compilation consistency checker
/usr/lib/pc2.*strings	text of the error messages
/usr/lib/how_pc	basic usage explanation
/usr/lib/libpc.a	intrinsic functions and I/O library
/usr/lib/libm.a	math library
/lib/libc.a	standard library, see *intro*(3)

SEE ALSO

Berkeley Pascal User's Manual
pi(1), pxp(1), pxref(1), sdb(1)

DIAGNOSTICS

For a basic explanation do

pc

See *pi*(1). for an explanation of the error message format. Internal errors cause messages containing the word SNARK.

AUTHORS

Charles B. Haley, William N. Joy, and Ken Thompson
Retargetted to the second pass of the portable *C* compiler by Peter Kessler
Runtime library and inline optimizer by M. Kirk McKusick
Separate compilation consistency checking by Louise Madrid

BUGS

The keyword **packed** is recognized but has no effect.

The binder is not as strict as described here, with regard to the rules about external declarations only in '.h' files and including '.h' files only at the outermost level. It will be made to perform these checks in its next incarnation, so users are warned not to be sloppy.

The **−z** flag doesn't work for separately compiled files.

Because the **−s** option is usurped by the compiler, it is not possible to pass the strip option to the loader. Thus programs which are to be stripped, must be run through *strip*(1) after they are compiled.

NAME

pc68 -- Pascal compiler for the MC68000

SYNOPSIS

pc68 [option] name ...

DESCRIPTION

Pc68 is the version of the portable Pascal* compiler that generates code for the MC68000. *Pc68* is a flexible program for translating between various types of files. The types catered for in order of appearance during translation are '.p' (Pascal source files), '.a68' or '.s' (assembly language files), '.b' (relocatable binary files), 'b.out' (absolute binary files), '.r' (byte-reversed files, cf. *rev68(1)*). and '.dl' (Macsbug download format, cf. *dl68(1)*).

Arguments to pc68 are either flags or input files. The type of an input file is normally determined by its suffix. When an argument to pc68 is not a flag and has none of the above suffixes, it is assumed to be of one of the types '.p', '.b', or 'b.out', namely the latest of these three consistent with the type of the output (e.g. if the output type were '.s' or '.b' then the input would have to be '.p').

Translation proceeds as follows. Each '.p' and '.s' program is translated to a '.b' relocatable using upas68, ugen68, and as68 as necessary. Then all .b files including those produced by translation are link edited into the one file, called 'b.out'. If the only input file was a single '.p' program then the '.b' file is deleted, otherwise all '.b' files are preserved.

The amount of processing performed by cc68 may be decreased or increased with some of the options. The -S option takes translation no further than '.s' files, i.e. only upas68 and ugen68 are applied. The -c option takes translation up to '.b' files, omitting the link-editing and not deleting any '.b' files. The -d option goes beyond 'b.out' to produce a '.dl' file (using dl68) that may be downloaded by the Motorola MACSBUG monitor and the Sun1 monitor. The -r option similarly goes beyond 'b.out' to produce a '.r' file (using rev68) that may be loaded directly by 68000 code based on ld68. Both -d and -r may be used together.

The output may be named explicitly with the -o option; the output file's name should follow -o. Otherwise the name is 'b.out' in the normal case, or 'filename.dl' for the -d option, or 'filename.r' for the -r option, where 'filename' is the first '.p', '.a68', '.s', or '.b' file named as an input. If the input is not in any of those three categories, the names 'd.out' and 'r.out' are used respectively for -d and -r.

The version of the target machine may be given as the flag **-v***n* where *n* is the version.
-vm is "Version Macsbug."
-vV means to run under the Vkernal.
-vxV means to run under the experimental version of the Vkernel.

A complete list of options interpreted by pc68 follows:

#flag Pass *flag* to the compiler. See the SOURCE FLAGS section below.

-c Suppress loading and produce '.b' file(s) from source file(s).

-g Have the compiler produce additional symbol table information for *pcdb68* (not implemented).

-e entrypoint
Entrypoint specifies where to begin execution.

-o output
Name the final output file *output* instead of *b.out.*

-s Accept standard Pascal only; non-standard constructs cause warning diagnostics (not implemented -- see internally controlled options).

-v n Use the 'n' version of the runtime support.

-w Suppress warning messages (not implemented).

--x Suppress passing the '-x' flag to the loader, retaining local symbols.

-E Run only the preprocessor (not implemented).

-L Make an assembly listing in filename.ls for each file assembled.

-O Invoke an object-code improver (not implemented).

-R Preserve relocation information in b.out.

-S Compile the named program, and leave the assembler-language output on the corresponding file suffixed '.s'. (No '.b' is created.).

-T org
Org specifies in hexadecimal where to begin loading the program.

-V Show the various stages of the compilation by printing images of the processes forked off to perform the actual work of the compilation.

-U Save the ucode associated with filname.p in filename.u (and filename.z, depending on the -W option).

-W Invoke the global ucode-to-ucode optimizer. If -U option active, generates filename.z.

-P Save all intermediate files. Most useful in conjuntion with -V (so that it is possible to find the intermediates).

Other arguments are taken to be loader option arguments, perhaps libraries of *pc68* compatible routines.

SEPARATE COMPILATION

Object files created by other language processors may be loaded together with object files created by *pc68*. Calling conventions are as in C, with **var** parameters and arrays passed by address. Don't pass structures except by VAR (pointer) if you call C, since here pc68 and cc68 differ. As a convenience, string constants are followed by a zero byte, so that you can use them as C strings when calling C routines.

To refer to a subroutine defined in a separate module, it must be declared. This follows the same syntax as **forward** declarations, except that the keyword **FORWARD** is replaced by **EXTERN** .

A file of subroutines is similar to a program except that there is no main program, and the **program** statement at the beginning of the file is replaced by a statement:

MODULE modulename;

The 'end;' of the last function in the file is followed by a period - there is no main program block. The modulename will become significant in Pascal*. Note that in identifiers (such as subprogram names) upper case is changed to lower case, and the linker is asked to ignore case.

OPENING FILES

To open a file for both input and output, use the standard procedure REVISE, which is analogous with RESET and REWRITE. NOT TESTED.

You can read and write files on machines which run a Leaf server. To open a file for reading do:

```
reset(file,'[hostname:username:password]filename');
```

The same syntax applies to rewrite. You can of course also use a Pascal string variable. Terminating spaces in hostname, username and password are ignored. (This should make it easier for a program to construct the appropriate filename string.)

You can leave out fields (or the entire second parameter), and the program will assume you want the same as before. If there is no "before", it will ask you.

Reset, Rewrite and Revise may have an optional third parameter, which is a string of switches. E.g.:

```
Reset(Input,'data1.txt','Nofilter;Prompt:"Try again!"');
```

Standard switches are:

- Prompt: The string is used as a prompt (interactive systems only). If a file name is NOT given, this prompt is used to get the file name from the user. If one IS given (like in the example above), the prompt is used to get another file name from the user if the file can't be opened.
- Default: The string is used are used as a default file name, which is used if the user types a carriage return in response to the prompt.
- Standard: If Reset, the standard input file is used. If Rewrite, the standard output is used.
- Nofilter: (Reset, Revise only.) Normally a text file is 'filtered' by the runtimes so that it conforms to the standard Pascal definition of a text file. Most notably, any end-of-line characters are changed into one space. The inclusion of Nofilter causes all characters to be passed through exactly as they appear in the text file. Eoln, Eopage and Readln still work as for standard files.

EXTENSIONS TO READ AND WRITE

For all field widths (if there are two field-width-type parameters, the first one only), a negative value will mean that the value written will be left-aligned instead of right-aligned. For string variables, if Abs(Fieldwidth) < Length, then the last Length - Abs(Fieldwidth) characters of the string will be written.

Variables of enumerated types may be read and written. The field width is interpreted the same as for strings. Enumerated constant names are uppercased when they are read in.

Sets of readable and writeable types may also be read and written. They appear exactly as set constants appear in Pascal programs. The field width is interpreted for each element the same as it would be for the set element type.

Integers may be written in other bases beside base 10 by including an optional field-width-type parameter, which may be anywhere from 1..16. The field width is the same as for base 10. Integers may also be read from a file in other than base 10, by including a field-width-type parameter in the call to Read or Readln.

Real numbers may have a capital "E" as well as the standard small "e" in the exponent part.

MORE ABOUT INPUT-OUTPUT

Lazy lokahead is used for text files, so that terminal input works reasonably.

The procedure Eopage is true iff a page marker has just been read, and the corresponding space in now in the file buffer.

Random-access in files is done with the standard procedure

```
seek (File, N);
```

This positions the file so that the next read/write will apply to component no. N of the file.

To close a file immediately do: close(file);

Function Filesize (var Filevar: Anyfile): 0..Maxint returns the current number of components in a file.

Function Curpos (var Filevar: Anyfile): 0..Maxint: Returns the current file position.

Procedure Filepos (var Filevar: Text; var Pagenum, Linenum, Charnum: 0..Maxint): Returns page, line number, and column number of the next character that will be read from the file (must be open for input). Does not work for random access.

TIME AND DATE ROUTINES

Clock -returns milli-seconds since the monitor was booted.

The following routines routines don't work if you want to run stand-alone, but need an operating system (V or Unix).

Ptime -returns (in theory) milli-seconds since midnight.
(under V, actually returns seconds*1000)

Pdate(day, month, year) -set day, month & year (say 1982).

Time(string) -sets string to 'HH:MM:SS'

Date(string) -sets string to 'MM/DD/YY'.

For both time and date, the string is a packed array [1..n] of char, where n>=8. (Any overflow is set to spaces.)

OTHER EXTENSIONS AND FEATURES

An "others" label in as CASE statement, indicates a default case.

To include a file as part of the program source do:
INCLUDE 'filename';
This is especially useful for declarations for seperately compiled modules.

Records declared as "packed" will be packed down to individual bits; however elements of packed arrays are at least a byte.

Function Min (X,Y: T): T -- returns the minimum of two arguments, which may be of any ordinal or real type.

Function Max (X,Y: T): T -- returns the maximum of two arguments.

Procedure Halt (Exitcode: Integer): Causes abnormal termination of a program. Passes a system-dependent exit code to the operating system.

The comment pairs '{ }' and '(* *) match independently, allowing limited nesting of comments.

SOURCE FLAGS

These flags can be passed to the compiler either at the command level when invoking pc68, or as comments within the program. A sample option line is a comment with # as its first character:
Sample option line: (*#g+,tdpy 1,tchk 1,U-8 *)
WARNING: Only (* *)-style comments will work; {#...} is ignored!

Sample command line: pc68 file.p #g:+ #tdpy:1 #tchk:1 #U:-8

Switch	Meaning (Note that the default value is shown)
B+	Bounds and nil pointer checking
C+	Print ucode
D-	Load with debugger
E+	Emit source code (for system debugging)
G-	Write error messages only to listing file
L-	Write full listing
I16	Number of characters of identifiers that are considered significant
O-	Emit optimizer-compatible code
P-	Keep execution profile
R0	Put up to N local variables in (data) registers (Register allocation should be done by the optimizer.)
S-	Accept standard Pascal only
T---	Code generator options
U+	Leave procedure names exactly as is
V32	Number of bits (16 or 32) to allocate for 'Integer'.
Wn	PRINT WARNINGS FOR:
W1	unused variables, types, procs, etc.
W10	nested comments
Z---	Optimizer switches

FILES

file.p	pascal source files
file.b	binary files
file.a68	assembler files
file.s	assembler files
file.ls	assembler listing
file.err	pascal listing

BUGS

Displacements off a frame pointer is limited to 16 bits signed, so very large locally-defined arrays will crash.

Some attempted bogus conversions (e.g. structure to real) aren't detected by the front end, and result in messages about 'Illegal CVT datatypes' from the code generator.

Sometimes formfeeds in the source get passed to the assembler, causing it to crash.

There is no macro processor.

GRIPES

Complaints should be sent to:
Per Bothner (mail to bothner@score)

There is also a pc68 mailing list. To add yourself to it, send a message to mailer@su-whitney. The first line of the message body should say:
add me to pc68

To say messages to to list, mail to pc68 at shasta, diablo, navajo or whitney.

NAME

pdx − pascal debugger

SYNOPSIS

pdx [−r] [*objfile*]

DESCRIPTION

Pdx is a tool for source level debugging and execution of Pascal programs. The *objfile* is an object file produced by the Pascal translator *pi*(1). If no *objfile* is specified, *pdx* looks for a file named "obj" in the current directory. The object file contains a symbol table which includes the name of the all the source files translated by *pi* to create it. These files are available for perusal while using the debugger.

If the file ".pdxinit" exists in the current directory, then the debugger commands in it are executed.

The **−r** option causes the *objfile* to be executed immediately; if it terminates successfully *pdx* exits. Otherwise it reports the reason for termination and offers the user the option of entering the debugger or simply letting *px* continue with a traceback. If **−r** is not specified, *pdx* just prompts and waits for a command.

The commands are:

run [*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.

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 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 Pascal boolean expression and is evaluated prior to printing the tracing information; if it is false then the information is not printed.

There is no restriction on the amount of information that can be traced.

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.

delete *command-number*
The trace or stop corresponding to the given number is removed. The numbers associated with traces and stops are printed by the **status** command.

status [> *filename*]
Print out the currently active **trace** and **stop** commands.

cont Continue execution from where it stopped. This can only be done when the program was stopped by an interrupt or through use of the **stop** command.

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.

print *expression* [, *expression* ...]
Print out the values of the Pascal expressions. Variables declared in an outer block but having the same identifier as one in the current block may be referenced as "*block-name* . *variable*".

whatis *identifier*
Print the declaration of the given identifier.

which *identifier*
Print the full qualification of the given identifer, i.e. the outer blocks that the identifier is associated with.

assign *variable expression*
Assign the value of the expression to the variable.

call *procedure(parameters)*
Execute the object code associated with the named procedure or function.

help Print out a synopsis of *pdx* commands.

gripe Invokes a mail program to send a message to the person in charge of *pdx*.

where Print out a list of the active procedures and functions and the respective source line where they are called.

source *filename*
Read *pdx* commands from the given *filename*. Especially useful when the *filename* has been created by redirecting a **status** command from an earlier debugging session.

dump [> *filename*]
Print the names and values of all active data.

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. As in the editor "$" can be used to refer to the last line. If no lines are specified, the entire file is 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.

file [*filename*]
Change the current source file name to *filename*. If none is specified then the current source file name is printed.

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.

pi Recompile the program and read in the new symbol table information.

sh *command-line*
Pass the command line to the shell for execution. The SHELL environment variable determines which shell is used.

alias *new-command-name old-command-name*
This command makes *pdx* respond to *new-command-name* the way it used to respond to *old-command-name*.

quit Exit *pdx*.

The following commands deal with the program at the *px* instruction level rather than source level. They are not intended for general use.

tracei [*address*] [**if** *cond*]
tracei [*variable*] [**at** *address*] [**if** *cond*]
stopi [*address*] [**if** *cond*]
stopi [**at**] [*address*] [**if** *cond*]
Turn on tracing or set a stop using a *px* machine instruction addresses.

xi *address* [, *address*]
Print the instructions starting at the first *address*. Instructions up to the second *address* are printed.

xd *address* [, *address*]
Print in octal the specified data location(s).

FILES

obj	Pascal object file
.pdxinit	*Pdx* initialization file

SEE ALSO

pi(1), px(1)
An Introduction to Pdx

BUGS

Pdx does not understand sets, and provides no information about files.

The *whatis* command doesn't quite work for variant records.

Bad things will happen if a procedure invoked with the **call** command does a non-local goto.

The commands **step** and **next** should be able to take a *count* that specifies how many lines to execute.

There should be commands **stepi** and **nexti** that correspond to **step** and **next** but work at the instruction level.

There should be a way to get an address associated with a line number, procedure or function, and variable.

Most of the command names are too long.

The alias facility is quite weak.

A *csh*-like history capability would improve the situation.

NAME

pi − Pascal interpreter code translator

SYNOPSIS

pi [option] [−i name ... **]** name.p

DESCRIPTION

Pi translates the program in the file *name.p* leaving interpreter code in the file *obj* in the current directory. The interpreter code can be executed using *px*. *Pix* performs the functions of *pi* and *px* for 'load and go' Pascal.

The following flags are interpreted by *pi;* the associated options can also be controlled in comments within the program as described in the *Berkeley Pascal User's Manual.*

−b Block buffer the file *output.*

−i Enable the listing for any specified procedures and functions and while processing any specified **include** files.

−l Make a program listing during translation.

−n Begin each listed **include** file on a new page with a banner line.

−p Suppress the post-mortem control flow backtrace if an error occurs; suppress statement limit counting.

−s Accept standard Pascal only; non-standard constructs cause warning diagnostics.

−t Suppress runtime tests of subrange variables and treat **assert** statements as comments.

−u Card image mode; only the first 72 characters of input lines are used.

−w Suppress warning diagnostics.

−z Allow execution profiling with *pxp* by generating statement counters, and arranging for the creation of the profile data file *pmon.out* when the resulting object is executed.

FILES

file.p	input file
file.i	**include** file(s)
/usr/lib/pi2.*strings	text of the error messages
/usr/lib/how_pi*	basic usage explanation
obj	interpreter code output

SEE ALSO

Berkeley Pascal User's Manual
pix(1), px(1), pxp(1), pxref(1)

DIAGNOSTICS

For a basic explanation do

pi

In the diagnostic output of the translator, lines containing syntax errors are listed with a flag indicating the point of error. Diagnostic messages indicate the action which the recovery mechanism took in order to be able to continue parsing. Some diagnostics indicate only that the input is 'malformed.' This occurs if the recovery can find no simple correction to make the input syntactically valid.

Semantic error diagnostics indicate a line in the source text near the point of error. Some errors evoke more than one diagnostic to help pinpoint the error; the follow-up messages begin with an ellipsis '...'.

The first character of each error message indicates its class:

E	Fatal error; no code will be generated.
e	Non-fatal error.
w	Warning — a potential problem.
s	Non-standard Pascal construct warning.

If a severe error occurs which inhibits further processing, the translator will give a diagnostic and then 'QUIT'.

AUTHORS

Charles B. Haley, William N. Joy, and Ken Thompson
Ported to VAX-11 by Peter Kessler

BUGS

The keyword **packed** is recognized but has no effect.

For clarity, semantic errors should be flagged at an appropriate place in the source text, and multiple instances of the 'same' semantic error should be summarized at the end of a **procedure** or **function** rather than evoking many diagnostics.

When **include** files are present, diagnostics relating to the last procedure in one file may appear after the beginning of the listing of the next.

NAME

pick – select messages by content

SYNOPSIS

pick { –cc | –date | –from | –search | –subject | –to | ––component } pattern [–src +folder] [msgs] [–help] [–scan] [–noscan] [–show] [–noshow] [–nofile] [–nokeep] [–file [–preserve] [–link] +folder ... [–nopreserve] [–nolink] [–keep [–stay] [–nostay] [+folder ...]]

typically:

pick –from jones –scan
pick –to holloway
pick –subject ned –scan –keep

DESCRIPTION

Pick searches messages within a folder for the specified contents, then performs several operations on the selected messages.

A modified *grep*(1) is used to perform the searching, so the full regular expression (see *ed*(1)) facility is available within 'pattern'. With '–search', pattern is used directly, and with the others, the grep pattern constructed is:

"^component:pattern"

This means that the pattern specified for a '–search' will be found everywhere in the message, including the header and the body, while the other search requests are limited to the single specified component. The expression '––component pattern' is a shorthand for specifying '–search "component:pattern" '; it is used to pick a component not in the set [cc date from subject to]. An example is "pick ––reply–to pooh –show".

Searching is performed on a per-line basis. Within the header of the message, each component is treated as one long line, but in the body, each line is separate. Lower-case letters in the search pattern will match either lower or upper case in the message, while upper case will match only upper case.

Once the search has been performed, the selected messages are scanned (see *scan*) if the '–scan' switch is given, and then they are shown (see *show*) if the '–show' switch is given. After these two operations, the file operations (if requested) are performed.

The '–file' switch operates exactly like the *file* command, with the same meaning for the '–preserve' and '–link' switches.

The '–keep' switch is similar to '–file', but it produces a folder that is a sub-folder of the folder being searched and defines it as the current folder (unless the '–stay' flag is used). This subfolder contains the messages which matched the search criteria. All of the MH commands may be used with the sub-folder as the current folder. This gives the user considerable power in dealing with sub-sets of messages in a folder.

The messages in a folder produced by '–keep' will always have the same numbers as they have in the source folder (i.e., the '–preserve' switch is automatic). This way, the message numbers are consistent with the folder from

which the messages were selected. Messages are not removed from the source folder (i.e., the '–link' switch is assumed). If a '+folder' is not specified, the standard name "select" will be used. (This is the meaning of "(select)" when it appears in the output of the *folder* command.) If '+folder' arguments are given to '–keep', they will be used rather than "select" for the names of the subfolders. This allows for several subfolders to be maintained concurrently.

When a '–keep' is performed, the subfolder becomes the current folder. This can be overridden by use of the '–stay' switch.

Here's an example:

```
 1 % folder +inbox
 2       inbox+ has  16 messages (  3– 22); cur=  3.
 3 % pick –from dcrocker
 4 6 hits.
 5 [+inbox/select now current]
 6 % folder
 7  inbox/select+ has   6 messages (  3– 16); cur=  3.
 8 % scan
 9    3+  6/20  Dcrocker        Re: ned file update issue...
10    6   6/23  Dcrocker        removal of files from /tm...
11    8   6/27  Dcrocker        Problems with the new ned...
12   13   6/28  dcrocker        newest nned I would ap...
13   15   7/ 5  Dcrocker        nned Last week I asked...
14   16   7/ 5  dcrocker        message id format I re...
15 % show all | print
16   [produce a full listing of this set of messages on the line printer.]
17 % folder –up
18       inbox+ has  16 messages (  3– 22); cur=  3; (select).
19 % folder –down
20  inbox/select+ has   6 messages (  3– 16); cur=  3.
21 % rmf
22 [+inbox now current]
23 % folder
24       inbox+ has  16 messages (  3– 22); cur=  3.
```

This is a rather lengthy example, but it shows the power of the MH package. In item 1, the current folder is set to inbox. In 3, all of the messages from dcrocker are found in inbox and linked into the folder "inbox/select". (Since no action switch is specified, '–keep' is assumed.) Items 6 and 7 show that this subfolder is now the current folder. Items 8 through 14 are a *scan* of the selected messages (note that they are all from dcrocker and are all in upper and lower case). Item 15 lists all of the messages to the high-speed printer. Item 17 directs *folder* to set the current folder to the parent of the selection-list folder, which is now current. Item 18 shows that this has been done. Item 19 resets the current folder to the selection list, and 21 removes the selection-list folder and resets the current folder to the parent folder, as shown in 22 and 23.

FILES

$HOME/mh_profile The user profile

PROFILE COMPONENTS

Path: To determine the user's MH directory
Folder–Protect: For protection on new folders

Current-Folder: To find the default current folder

DEFAULTS

'–src +folder' defaults to current
'msgs' defaults to all
'–keep +select' is the default if no '–scan', '–show', or '–file' is specified

CONTEXT

If a '–src +folder' is specified, it will become the current folder, unless a '–keep' with 0 or 1 folder arguments makes the selection-list subfolder the current folder. Each selection-list folder will have its current message set to the first of the messages linked into it unless the selection list already existed, in which case the current message won't be changed.

NAME

ping – IP/ICMP echo user program

SYNOPSIS

ping [–cnnn **] [** –snnn **]** hostname **[ ]**

DESCRIPTION

Ping is used to send an ICMP (Internet Control Message Protocol) "Echo Me" packet to a host; it waits for a reply to see if the host responds. Since every IP host is required to respond to ICMP packets, this is a simple way to determine if a host is up.

If more than one host name argument is given, the hosts are pinged in order.

OPTIONS

The following options are recognized. Note that numeric arguments follow the option flags immediately, without intervening spaces.

–cnnn	For each host specified, send the echo *nnn* times.
–snnn	Make the packets *nnn* bytes long.

BUGS

By changing the default length you may create a situation where Unix may send the echo packet but will drop the response, thus confusing the issue.

Since the Unix hostname software is abysmally slow, it often takes longer to look up the hostname than it does to exchange packets.

NAME

pix − Pascal interpreter and executor

SYNOPSIS

pix [**−blnpstuwz**] [**−i** name ...] name.p [argument ...]

DESCRIPTION

Pix is a 'load and go' version of Pascal which combines the functions of the interpreter code translator *pi* and the executor *px*. It uses *pi* to translate the program in the file *name.p* and, if there were no fatal errors during translation, causes the resulting interpreter code to be executed by *px* with the specified arguments. A temporary file is used for the object code; the file *obj* is neither created nor destroyed.

FILES

/usr/ucb/pi	Pascal translator
/usr/ucb/px	Pascal executor
/tmp/pix*	temporary
/usr/lib/how_pix	basic explanation

SEE ALSO

Berkeley Pascal User's Manual
pi(1), px(1)

DIAGNOSTICS

For a basic explanation do

pix

AUTHORS

Susan L. Graham and William N. Joy

NAME

plot – graphics filters

SYNOPSIS

plot [–Tterminal [raster]]

DESCRIPTION

These commands read plotting instructions (see *plot*(5)) from the standard input, and in general produce plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* type is specified, the environment parameter $TERM (see *environ*(5)) is used. Known *terminals* are:

4014 Tektronix 4014 storage scope.

450 DASI Hyterm 450 terminal (Diablo mechanism).

300 DASI 300 or GSI terminal (Diablo mechanism).

300S DASI 300S terminal (Diablo mechanism).

ver Versatec D1200A printer-plotter. This version of *plot* 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 optional argument causes a previously scan-converted file *raster* to be sent to the plotter.

var Varian plotter; similar to versatec except that data is buffered internally, /usr/tmp/raster is not written, and it cannot dump a scan-converted file. However, it's a lot faster.

tek4025,4025
Tektronix 4025 terminal

hp2648,2648
Hewlett-Packard 2648 terminal

c100, c100rv, C100, C100rv
Concept 100 (for light duty low accuracy work only)

press Sends a plot to a Pressfile printer.

dover synonym for "press"

sungr Per Bothner's Sun Graphics Terminal (probably obsolete)

FILES

/usr/bin/tek
/usr/bin/t450
/usr/bin/t300
/usr/bin/t300s
/usr/bin/vplot
/usr/bin/varplot
/usr/bin/pressplot
/usr/bin/tek4025
/usr/bin/hp2648
/usr/bin/concept
/usr/bin/sunplot
/usr/tmp/raster

SEE ALSO

plot(3), plot(5)

BUGS

There is no lockout protection for /usr/tmp/raster.

NAME

pmerge — pascal file merger

SYNOPSIS

pmerge name.p ...

DESCRIPTION

Pmerge assembles the named Pascal files into a single standard Pascal program. The resulting program is listed on the standard output. It is intended to be used to merge a collection of separately compiled modules so that they can be run through **pi** , or exported to other sites.

FILES

/usr/tmp/MG* default temporary files

SEE ALSO

pc(1), pi(1),
Auxiliary documentation *Berkeley Pascal User's Manual.*

AUTHOR

M. Kirk McKusick

BUGS

Very minimal error checking is done, so incorrect programs will produce unpredictable results. Block comments should be placed after the keyword to which they refer or they are likely to end up in bizarre places.

NAME

postnews – submit news articles

SYNOPSIS

postnews [*article*]

DESCRIPTION

Postnews is a shell script that calls *inews*(1) to submit news articles to USENET. It will prompt the user for the title of the article (which should be a phrase suggesting the subject, so that persons reading the news can tell if they are interested in the article) for the newsgroup, and for the distribution.

An omitted newsgroup (from hitting return) will default to *general*.

general is read by everyone on the local machine. Other possible newsgroups include, but are not limited to, *btl.general*, which is read by all users at all Bell Labs sites on USENET, *net.general*, which is read by all users at all sites on USENET, and *net.news*, which is read by users interested in the network news on all sites. There is often a local set of newsgroups, such as *ucb.all*, that circulate within a local set of machines. (In this case, *ucb* newsgroups circulate among machines at the University of California at Berkeley.)

The distribution can be any valid newsgroup name list, and defaults to the same as the newsgroup. (If they are the same, the distribution will be omitted from the headers put into the editor buffer.) A distribution header will, if given, be included in the headers of the article, affecting where the article is distributed to.

After entering the title, newsgroup, and distribution, the user will be placed in an editor. If $EDITOR is set in the environment, that editor will be used. Otherwise, postnews defaults to *vi*(1).

An initial set of headers containing the subject and newsgroups will be placed in the editor, followed by a blank line. The article should be appended to the buffer, after the blank line. These headers can be changed, or additional headers added, while in the editor, if desired.

Optionally, the article will be read from the specified *filename*.

For more sophisticated uses, such as posting news from a program, see *inews*(1).

SEE ALSO

Mail(1), checknews(1), inews(1), mail(1), readnews(1).

NAME

pr − print file

SYNOPSIS

pr [option] ... [file] ...

DESCRIPTION

Pr produces a printed listing of one or more *files*. The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. If there are no file arguments, *pr* prints its standard input.

Options apply to all following files but may be reset between files:

−*n* Produce *n*-column output.

+*n* Begin printing with page *n*.

−h Take the next argument as a page header.

−w*n* For purposes of multi-column output, take the width of the page to be *n* characters instead of the default 72.

−f Use formfeeds instead of newlines to separate pages. A formfeed is assumed to use up two blank lines at the top of a page. (Thus this option does not affect the effective page length.)

−l*n* Take the length of the page to be *n* lines instead of the default 66.

−t Do not print the 5-line header or the 5-line trailer normally supplied for each page.

−s*c* Separate columns by the single character *c* instead of by the appropriate amount of white space. A missing *c* is taken to be a tab.

−m Print all *files* simultaneously, each in one column,

Inter-terminal messages via *write*(1) are forbidden during a *pr*.

FILES

/dev/tty? to suspend messages.

SEE ALSO

cat(1)

DIAGNOSTICS

There are no diagnostics when *pr* is printing on a terminal.

NAME

pr68 – print extended statistics on .b file

SYNOPSIS

pr68 file

DESCRIPTION

Pr68 prints the header information, symbol table, and relocation commands of a .b or .68 file. Verifies that the text and data segments are multiples of 4.

AUTHOR

C.J. Terman

NAME

pressimp – convert press files to ImPress format and print them on the ImPrint printer.

SYNOPSIS

pressimp [options] [files]

DESCRIPTION

Pressimp reads in Xerox press files, converts them to ImPress format and queues them to be printed on the ImPrint printer. If no input files are specified, standard input is used (although pipes cannot be used).

The environment variable IMPRESS may be used to specify defaults. The value of IMPRESS is parsed as a string of arguments before the arguments that appear on the command line.

The possible options are:

-b *banner* Uses *banner* to label the output. It will appear on the banner page on the line labeled "Description:".

-c *n* Causes *n* copies of the output to be produced. The default is one copy.

-n *name* Causes *name* to be used (the "For:" field on the cover sheet). This usually defaults to your full name, causing output to be filed by the first character of your first name. If you'd rather go by your last name, then use '-n "Bovik, Harold"'. You can arrange to have this done by default by using the IMPRESS environment variable.

-N Causes the ImPrint printer daemon to notify you when the job is done, by writing to your terminal.

-M Causes the ImPrint printer daemon to notify you when the job is done, by sending you mail.

-q Quiet mode; normally, *pressimp* gives page number and font information, but users who prefer silence can suppress all messages with this option.

-d Causes some debugging information to be printed out as font files are being read.

-D Causes very much debugging information to be printed out as font files are being read, as well as the position of every character.

-i *name* Causes the press file to be written to the named file rather than being shipped to the ImPrint printer.

ENVIRONMENT

IMPRESS	string of options to be used by *pressimp*.
TFM	the path name of a file to use in preference to /usr/local/fonts/tfm to find the font width information.
RAS	the path name of a directory to use in preference to /usr/local/fonts/raster to find the character rasters.

FILES

/usr/tmp/ImPressXXXXXX	default imPress file output.
/usr/local/fonts/tfm	describes all the available font families.
/usr/local/fonts/raster	describes all the available fonts magnifications.
/usr/local/lib/ipr	ImPrint printer Printer queuer.

SEE ALSO

ipq(1), ipr(1), iprm(1), itroff(1), imprint(1), scribe(1)

AUTHOR

Bill Nowicki, Stanford Univeristy.

DIAGNOSTICS

Lots, but they should be self explanatory.

BUGS

Inherits lots of bugs from other programs. It does not implement the entire Press specification, only those produced by common utilities like Scribe.

NAME

prev – show the previous message

SYNOPSIS

prev [+folder] [–switches for *l*] [–help]

DESCRIPTION

Prev performs a *show* on the previous message in the specified (or current) folder. Like *show*, it passes any switches on to the program *l*, which is called to list the message. This command is exactly equivalent to "show prev".

FILES

$HOME/mh_profile The user profile

PROFILE COMPONENTS

Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder

CONTEXT

If a folder is specified, it will become current, and the message that is shown (i.e., the previous message in sequence) will become the current message.

NAME

print − pr to the line printer

SYNOPSIS

print file ...

DESCRIPTION

Print pr's a copy of each named file on the line printer. It is a one line shell script:

```
lpr -p $*
```

SEE ALSO

lpr(1), pr(1)

NAME

printenv — print out the environment

SYNOPSIS

printenv [name]

DESCRIPTION

Printenv prints out the values of the variables in the environment. If a *name* is specified, only its value is printed.

If a *name* is specified and it is not defined in the environment, *printenv* returns exit status 1, else it returns status 0.

SEE ALSO

sh(1), environ(7), csh(1)

NAME

prmail — print out mail in the post office

SYNOPSIS

prmail [user ...]

DESCRIPTION

Prmail prints the mail which waits for you, or the specified user, in the post office. The mail is not disturbed.

FILES

/usr/spool/mail/* post office

SEE ALSO

biff(1), mail(1), from(1), binmail(1)

NAME

prof − display profile data

SYNOPSIS

prof [**−a**] [**−l**] [**−n**] [**−z**] [**−s**] [**−v** [**−***low* [**−***high*]]] [a.out [mon.out ...]]

DESCRIPTION

Prof interprets the file produced by the *monitor* subroutine. Under default modes, the symbol table in the named object file *(a.out* default) is read and correlated with the profile file *(mon.out* default). For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call. If more than one profile file is specified, the output represents the sum of the profiles.

In order for the number of calls to a routine to be tallied, the **−p** option of *cc, f77* or *pc* must have been given when the file containing the routine was compiled. This option also arranges for the profile file to be produced automatically.

Options are:

−a all symbols are reported rather than just external symbols.

−l the output is sorted by symbol value.

−n the output is sorted by number of calls

−s a summary profile file is produced in *mon.sum*. This is really only useful when more than one profile file is specified.

−v all printing is suppressed and a graphic version of the profile is produced on the standard output for display by the *plot*(1) filters. When plotting, the numbers *low* and *high*, by default 0 and 100, may be given to cause a selected percentage of the profile to be plotted with accordingly higher resolution.

−z routines which have zero usage (as indicated by call counts and accumulated time) are nevertheless printed in the output.

FILES

mon.out for profile
a.out for namelist
mon.sum for summary profile

SEE ALSO

monitor(3), profil(2), cc(1), plot(1G)

BUGS

Beware of quantization errors.

Is confused by *f77* which puts the entry points at the bottom of subroutines and functions.

NAME

prompter – prompting editor front end

SYNOPSIS

prompter [–erase chr] [–kill chr] [–help]

DESCRIPTION

This program is not called directly but takes the place of an editor and acts as an editor front end. *Prompter* is an editor which allows rapid composition of messages. It is particularly useful to network and low-speed (less than 2400 baud) users of MH. It is an MH program in that it can have its own profile entry with switches, but it can't be invoked directly as all other MH commands can; it is an editor in that it is invoked by an "–editor prompter" switch or by the profile entry "Editor: prompter", but functionally it is merely a text-collector and not a true editor.

Prompter expects to be called from *comp*, *repl*, *dist*, or *forw*, with a draft file as an argument. For example, "comp –editor prompter" will call *prompter* with the file "draft" already set up with blank components. For each blank component it finds in the draft, it prompts the user and accepts a response. A <RETURN> will cause the whole component to be left out. A "\" preceding a <RETURN> will continue the response on the next line, allowing for multiline components.

Any component that is non-blank will be copied and echoed to the terminal.

The start of the message body is prompted by a line of dashes. If the body is non-blank, the prompt is "------Enter additional text". Message-body typing is terminated with a <CTRL-D> (or <OPEN>). Control is returned to the calling program, where the user is asked "What now?". See *comp* for the valid options.

The line editing characters for kill and erase may be specified by the user via the arguments "–kill chr" and "–erase chr", where chr may be a character; or "\nnn", where nnn is the octal value for the character. (Again, these may come from the default switches specified in the user's profile.)

A <DEL> during message-body typing is equivalent to <CTRL-D> for compatibility with NED. A <DEL> during component typing will abort the command that invoked *prompter*.

PROFILE COMPONENTS

prompter-next: editor to be used on exit from *prompter*

NAME

ps – process status

SYNOPSIS

ps [acegklstuvwx#]

DESCRIPTION

Ps prints information about processes. Normally, only your processes are candidates to be printed by *ps;* specifying **a** causes other users processes to be candidates to be printed; specifying **x** includes processes without control terminals in the candidate pool.

All output formats include, for each process, the process id PID, control terminal of the process TT, cpu time used by the process TIME (this includes both user and system time), the state STAT of the process, and an indication of the COMMAND which is running. The state is given by a sequence of four letters, e.g. "RWNA". The first letter indicates the runnability of the process: R for runnable processes, T for stopped processes, P for processes in page wait, D for those in disk (or other short term) waits, S for those sleeping for less than about 20 seconds, and I for idle (sleeping longer than about 20 seconds) processes. The second letter indicates whether a process is swapped out, showing W if it is, or a blank if it is loaded (in-core); a process which has specified a soft limit on memory requirements and which is exceeding that limit shows >; such a process is (necessarily) not swapped. The third letter indicates whether a process is running with altered CPU scheduling priority (nice); if the process priority is reduced, an N is shown, if the process priority has been artificially raised then a '<' is shown; processes running without special treatment have just a blank. The final letter indicates any special treatment of the process for virtual memory replacement; the letters correspond to options to the *vadvise*(2) call; currently the possibilities are A standing for VA_ANOM, S for VA_SEQL and blank for VA_NORM; an A typically represents a *lisp*(1) in garbage collection, S is typical of large image processing programs which are using virtual memory to sequentially address voluminous data.

Here are the options:

a asks for information about all processes with terminals (ordinarily only one's own processes are displayed).

c prints the command name, as stored internally in the system for purposes of accounting, rather than the command arguments, which are kept in the process' address space. This is more reliable, if less informative, since the process is free to destroy the latter information.

e Asks for the environment to be printed as well as the arguments to the command.

g Asks for all processes. Without this option, *ps* only prints "interesting" processes. Processes are deemed to be uninteresting if they are process group leaders. This normally eliminates top-level command interpreters and processes waiting for users to login on free terminals.

k causes the file */vmcore* is used in place of */dev/kmem* and */dev/mem*. This is used for post-mortem system debugging.

l asks for a long listing, with fields PPID, CP, PRI, NI, ADDR, SIZE, RSS and WCHAN as described below.

s Adds the size SSIZ of the kernel stack of each process (for use by system maintainers) to the basic output format.

t*x* restricts output to processes whose controlling tty is *x* (which should be specified as printed by *ps*, e.g. *t3* for tty3, *tco* for console, *td0* for ttyd0, *t?* for processes with no tty, *t* for processes at the current tty, etc). This option must be the last one given.

u A user oriented output is produced. This includes fields USER, %CPU, NICE, SIZE, and

RSS as described below.

v A version of the output containing virtual memory statistics is output. This includes fields RE, SL, PAGEIN, SIZE, RSS, LIM, TSIZ, TRS, %CPU and %MEM, described below.

w Use a wide output format (132 columns rather than 80); if repeated, e.g. ww, use arbitrarily wide output. This information is used to decide how much of long commands to print.

x asks even about processes with no terminal.

A process number may be given, (indicated here by #), in which case the output is restricted to that process. This option must also be last.

A second argument tells *ps* where to look for *core* if the **k** option is given, instead of /vmcore. A third argument is the name of a swap file to use instead of the default /dev/drum. If a fourth argument is given, it is taken to be the file containing the system's namelist. Otherwise, /vmunix is used.

Fields which are not common to all output formats:

USER	name of the owner of the process
%CPU	cpu utilization of the process; this is a decaying average over up to a minute of previous (real) time. Since the time base over which this is computed varies (since processes may be very young) it is possible for the sum of all %CPU fields to exceed 100%.
NICE	(or NI) process scheduling increment (see *setpriority*(2))
SIZE	virtual size of the process (in 1024 byte units)
RSS	real memory (resident set) size of the process (in 1024 byte units)
LIM	soft limit on memory used, specified via a call to *setrlimit*(2); if no limit has been specified then shown as *xx*
TSIZ	size of text (shared program) image
TRS	size of resident (real memory) set of text
%MEM	percentage of real memory used by this process.
RE	residency time of the process (seconds in core)
SL	sleep time of the process (seconds blocked)
PAGEIN	number of disk i/o's resulting from references by the process to pages not loaded in core.
UID	numerical user-id of process owner
PPID	numerical id of parent of process
CP	short-term cpu utilization factor (used in scheduling)
PRI	process priority (non-positive when in non-interruptible wait)
ADDR	swap address of the process
WCHAN	event on which process is waiting (an address in the system), with the initial part of the address trimmed off e.g. 80004000 prints as 4000.

F flags associated with process as in *<sys/proc.h>*:

SLOAD	000001	in core
SSYS	000002	swapper or pager process
SLOCK	000004	process being swapped out
SSWAP	000008	save area flag
STRC	000010	process is being traced
SWTED	000020	another tracing flag
SULOCK	000040	user settable lock in core
SPAGE	000080	process in page wait state
SKEEP	000100	another flag to prevent swap out

SDLYU	000200	delayed unlock of pages
SWEXIT	000400	working on exiting
SPHYSIO	000800	doing physical i/o (bio.c)
SVFORK	001000	process resulted from vfork()
SVFDONE	002000	another vfork flag
SNOVM	004000	no vm, parent in a vfork()
SPAGI	008000	init data space on demand from inode
SANOM	010000	system detected anomalous vm behavior
SUANOM	020000	user warned of anomalous vm behavior
STIMO	040000	timing out during sleep
SDETACH	080000	detached inherited by init
SOUSIG	100000	using old signal mechanism

A process that has exited and has a parent, but has not yet been waited for by the parent is marked <defunct>; a process which is blocked trying to exit is marked <exiting>; *Ps* makes an educated guess as to the file name and arguments given when the process was created by examining memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

FILES

/vmunix	system namelist
/dev/kmem	kernel memory
/dev/drum	swap device
/vmcore	core file
/dev	searched to find swap device and tty names

SEE ALSO

kill(1), w(1)

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality.

NAME

pti − phototypesetter interpreter

SYNOPSIS

pti [file ...]

DESCRIPTION

Pti shows the commands in a stream from the standard output of *troff*(1) using *troff's* **−t** option, interpreting them as they would act on the typesetter. Horizontal motions shows as counts in internal units and are marked with '<' and '>' indicating left and right motion. Vertical space is called *lead* and is also indicated.

SEE ALSO

troff(1)

BUGS

Too cryptic for normal users, who should use "troff −a ...".

NAME

ptx − permuted index

SYNOPSIS

ptx [option] ... [input [output]]

DESCRIPTION

Ptx generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *Ptx* produces output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx may be an *nroff* or *troff*(1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head*, at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

The following options can be applied:

−f Fold upper and lower case letters for sorting.

−t Prepare the output for the phototypesetter; the default line length is 100 characters.

−w *n* Use the next argument, *n*, as the width of the output line. The default line length is 72 characters.

−g *n* Use the next argument, *n*, as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 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. In any case, 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 5th field on each output line.

The index for this manual was generated using *ptx*.

FILES

/usr/bin/sort
/usr/lib/eign

BUGS

Line length counts do not account for overstriking or proportional spacing.

NAME

pupecho, echoserve – Pup Echo protocol user and server

SYNOPSIS

pupecho [-v] host [string]

echoserve [-c] [-v]

DESCRIPTION

The Pup Echo Protocol is a simple protocol used for determining that Pup packets can be exchanged between two hosts. The *user* sends a packet with PupType ECHOME to the ECHOSERVER socket, and the server responds with an IAMECHO packet, containing the same PupData as the received packet.

The server can respond with an IAMBADECHO if the received packet appeared damaged, an ERRORPUP if it doesn't like to echo, or simply not respond at all.

To use *pupecho*, invoke it with the name of a host (that you believe will echo packets) and a string to send as data (this is fairly arbitrary.) The program will send packets at short intervals, until it receives an interrupt signal (normally Ctrl/C), at which point it will print out statistics showing how successful it was. The -v option will print out the status of every packet.

To use *echoserve*, invoke with –c, if you want it to check the checksums on incoming packets, or without –c, if you want it to ignore bad checksums. It will echo to any host sending it an ECHOME.

SEE ALSO

Xerox document, *Pup Echo Protocol*

DIAGNOSTICS

cryptic

AUTHOR

Jeffrey Mogul, Slight mods by Bill Nowicki

NAME

pupftp — Pup File Transfer Program

SYNOPSIS

pupftp *[host]*

DESCRIPTION

Pupftp is a program used to transfer files to and from another host on a PUP network. It is quite similar to the FTP program used on the XEROX Altos. It can only be used to transfer to or from a host which is running a *pupftp server* program, such as the IFS, a Dolphin or Dandelion, or another Vax.

If you do not specify a host, it prompts you for the name of the server; you may give either a host name or an (octal) host number.

GENERAL NOTES

In general, you must be logged in (see the command descriptions below and *netalias(1)*) before doing anything at the remote host. Commands are read from standard input, which may be a terminal or a command file.

Files on the remote host are named according to that host's file-naming rules. Each operating system has a different concept of a "directory" and "name body", for example.

The interactive command parser of the *pupftp* program is similar to that of Alto FTP, the IFS, or TOPS-20; your 'erase' character deletes characters that you have typed; if you type the unambiguous prefix of a command, the program completes the command for you after you type a punctuation, space, or control character.

COMMANDS

assume — The **assume** command invokes a mode in which the files names are assumed to be identical on both the local and remote machines. For example, if you are doing a copy (or comparison) of all the files in a given directory, this mode can save you much typing. Subsequent **assume** commands invert the state of this mode (just like the **verbose** and **!** commands: see below).

change — The **change** command changes the current workng directory on the *local* machine. The corresponding command for changing the remote directory is **directory.**

compare — The **compare** command allows you to compare remote files to local ones. The remote file is piped into the *diff(1)* program (or the *cmp(1)* program for binary files)
and compared with the specified local file. The default local file is the remote file's name body.

delete — The **delete** command allows you to specify files to be deleted at the remote host; after you type the filename, you are shown some information about the file, and then must answer 'y' or 'n' to either delete it, or not.

directory — The **directory** command changes your default directory at the remote host. The directory name typed becomes the default directory until a new 'directory' or 'login' is executed. If a carriage return is hit the default of the remote user's directory is used. Entering "." (a single dot) sets the remote directory to the current working directory.

list — The **list** command allows you to find out about a file, or group of files, at the remote host. Since this may generate a lot of information, when the program has printed a page of data, it prompts you with the line "--Space for more--"; unfortunately there is not a simple way to interrupt this listing.

login — This command must be executed before doing anything else, unless your

password had been saved by the *netalias(1)* feature. The **login** command requires a username for the remote system, and then a password (which is not echoed.)

quit — The **quit** command is used to terminate a connection with the remote host; the connection is also terminated (although not as cleanly) if the process is killed.

retrieve — The **retrieve** command allow remote files to be retrieved. It prompts you for a local filename to put each retrieved file under. If you do not specify one, the default filename is the the remote base filename. If you specify – (a hyphen) as the file name, it will be sent to standard output.

shell — The **shell** command accepts a standard Unix shell command and executes it. For example, the **ls** command will list your local files, and **pwd** will print your current working directory.

store — The **store** command takes a local filename to be stored on the remote system; it prompts you for a remote filename. The default is the local filename with the directory information removed, placed in the directory of the remote account under which you are logged in. It then asks you whether the file should be stored as a Binary file if you have not set a default mode; the default is 'n' (i.e. Text).

type — This command allows you to specify the default file type to use. There are four possibilities. **Text** type transfers perform the proper conversions to and from Unix end of line conventions. **Binary** type transfers are done with 8 bit bytes, swapping the bytes in every word. This means that Press files, for example, can be sent between Vaxes and Altos with no problems. **Guess** type transfers (initially the default) will use the mode specifyed by the server. On **store** commands, the first 512 bytes of the file are examined. If any of these bytes are not 7 bit ASCII characters, the file is considered binary, otherwise it is considered as text. **Query** type transfers will use the type specified by the server on retrieves, but ask the user to specify the type of each file stored.

verbose — The **verbose** command turns on and off very long-winded debugging information. Used for maintenence only.

! — Turns on and off the printing of exclamation points on each buffer sucessfully transfered. Initially this feature is enabled, to allow a small amount of confidence that some progress is being made.

SEE ALSO

eftp(1), puptelnet(1), netalias(1), ftpser(8)

AUTHORS

Bill Nowicki, based on an earlier version by Dan Kolkowitz & Erik Hedberg

DIAGNOSTICS

Should be self-explanatory

BUGS

There should be a way of interrupting a **list** command, and aborting a **retrieve** request. Should expand wildcards on **store** commands.

NAME

puproute – print Pup network routing table information

SYNOPSIS

puproute [–a] [–c] [–s [interval]]

DESCRIPTION

Puproute is used to print the Pup routing table; this is mostly a debugging tool.

OPTIONS

–a	Normally, networks and gateways are listed by name, but giving this option lists them by address.
–c	Lists only directly-connected networks.
–s [interval]	If this option is given, *puproute* will repeat continually, sleeping *interval* seconds between repetitions. If *interval* is omitted, the period defaults to 30 seconds; this is the normal update interval on a quiescent network.

AUTHOR

Jeffrey Mogul

BUGS

Using *puproute* without the –a option can be slow, at least on the first time around, because translating the addresses to names takes a while. If the –s option is used, subsequent iterations are much faster, since a cache is kept of address-to-name translations.

NAME

puptelnet – connect your terminal to a remote computer via Pup network

SYNOPSIS

puptelnet [options] [host] [typescript-file]

DESCRIPTION

Puptelnet connects your terminal to a remote computer in such a way that you can more or less pretend you are connected to it directly. The remote computer must be connected to this Unix system by means of an appropriate Pup network or combination of networks. *Puptelnet* initially checks to see if the requested host is connected on the local Pup internet. If it is unable to find the requested host there, it then invokes the *telnet*(1) program to connect through IP/TCP protocols.

The *host* name that you type can be either a symbolic name or an octal ethernet port number. Octal port numbers are useful only in the emergency situation of the name server being broken.

If no host is specified as an argument, you will be prompted for one. If a typescript file is specified, all output is sent to that file in addition to being printed on the terminal.

The *puptelnet* program exits with the message [Connection Closed] when either the remote host or the user closes the connection.

The recognized options are:

-7 – open the connection in 7-bit character mode. This means that regardless of how many bits per character your terminal sends, only the rightmost seven will be transmitted to the distant host. This mode is necessary if you are using a terminal that does not generate proper parity bits (such as a C100) to connect to a host that insists on them (such as CMU or MIT).

-8 – open the connection in 8-bit character mode. This mode will be necessary if you are connecting to a machine with 8-bit characters (like a VAX or IFS) and you want to control the parity bits sent in the connection. If you are using a terminal with an EDIT or META key, for example, you will want to use 8-bit connections.

-d – wizards only. Run in BSP diagnostic mode.

Commands can be entered to *puptelnet* by typing the escape character, control up-arrow, which is the same character that the TOPS-20 *puptelnet* program uses, and a command character. The escape commands are:

↑↑ – (control up-arrow) send the escape character

c – close the connection

Note that to send *n* escape characters, 2 to the power *n* escape characters must be typed. The control up-arrow character is generated on the Tektronix 4025 and Alto keyboards by typing control shift 6, and on the hazeltine 1510 by control shift N.

AUTHOR

Bill Nowicki, also hacked by Jeff Mogul and (beware!!!) by Brian Reid.

SEE ALSO

The PUP protocol specifications (an internal Xerox memo), and the manual section describing the Unix PUP package.

DIAGNOSTICS

All puptelnet messages are enclosed in square brackets. Any abort messages sent by the remote host are simply printed and the connection closed. If *puptelnet* is unable to open the required two ethernet ports:

[Sorry, no ethernet ports available]

If the connection times out, i.e. the remote host does not respond to timing packets in about 30-40 seconds, the message

[Connection closed - Timeout]

is printed.

BUGS

Timeouts are very hard to implement correctly -- one must compromise between impatient users and slow systems.

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

SEE ALSO

cd(1), csh(1), getwd(3)

BUGS

In *csh*(1) the command *dirs* is always faster (although it can give a different answer in the rare case that the current directory or a containing directory was moved after the shell descended into it).

NAME

px – Pascal interpreter

SYNOPSIS

px [obj [argument ...]]

DESCRIPTION

Px interprets the abstract machine code generated by *pi*. The first argument is the file to be interpreted, and defaults to *obj*; remaining arguments are available to the Pascal program using the built-ins *argv* and *argc*. *Px* is also invoked by *pix* when running 'load and go'.

If the program terminates abnormally an error message and a control flow backtrace are printed. The number of statements executed and total execution time are printed after normal termination. The **p** option of *pi* suppresses all of this except the message indicating the cause of abnormal termination.

FILES

obj	default object file
pmon.out	profile data file

SEE ALSO

Berkeley Pascal User's Manual
pi(1), pix(1)

DIAGNOSTICS

Most run-time error messages are self-explanatory. Some of the more unusual ones are:

Reference to an inactive file
: A file other than *input* or *output* was used before a call to *reset* or *rewrite*.

Statement count limit exceeded
: The limit of 500,000 executed statements (which prevents excessive looping or recursion) has been exceeded.

Bad data found on integer read
Bad data found on real read
: Usually, non-numeric input was found for a number. For reals, Pascal requires digits before and after the decimal point so that numbers like '.1' or '21.' evoke the second diagnostic.

panic: *Some message*
: Indicates a internal inconsistency detected in *px* probably due to a Pascal system bug.

AUTHORS

Charles B. Haley, William Joy, and Ken Thompson
VAX-11 version by Kirk McKusick

BUGS

Post-mortem traceback is not limited; infinite recursion leads to almost infinite traceback.

NAME

pxp — Pascal execution profiler

SYNOPSIS

pxp [**−acdefjnstuw_**] [**−23456789**] [**−z** [name ...]] name.p

DESCRIPTION

Pxp can be used to obtain execution profiles of Pascal programs or as a pretty-printer. To produce an execution profile all that is necessary is to translate the program specifying the **z** option to *pi* or *pix*, to execute the program, and to then issue the command

pxp −z name.p

A reformatted listing is output if none of the **c**, **t**, or **z** options are specified; thus

pxp old.p > new.p

places a pretty-printed version of the program in 'old.p' in the file 'new.p'.

The use of the following options of *pxp* is discussed in sections 2.6, 5.4, 5.5 and 5.10 of the *Berkeley Pascal User's Manual*.

−a Print the bodies of all procedures and functions in the profile; even those which were never executed.

−c Extract profile data from the file *core*.

−d Include declaration parts in a profile.

−e Eliminate **include** directives when reformatting a file; the **include** is replaced by the reformatted contents of the specified file.

−f Fully parenthesize expressions.

−j Left justify all procedures and functions.

−n Eject a new page as each file is included; in profiles, print a blank line at the top of the page.

−s Strip comments from the input text.

−t Print a table summarizing **procedure** and **function** call counts.

−u Card image mode; only the first 72 characters of input lines are used.

−w Suppress warning diagnostics.

−z Generate an execution profile. If no *name*s, are given the profile is of the entire program. If a list of names is given, then only any specified **procedures** or **functions** and the contents of any specified **include** files will appear in the profile.

−_ Underline keywords.

−*d* With *d* a digit, $2 \leqslant d \leqslant 9$, causes *pxp* to use *d* spaces as the basic indenting unit. The default is 4.

FILES

name.p	input file
name.i	include file(s)
pmon.out	profile data
core	profile data source with **−c**
/usr/lib/how_pxp	information on basic usage

SEE ALSO

Berkeley Pascal User's Manual
pi(1), px(1)

DIAGNOSTICS

For a basic explanation do

pxp

Error diagnostics include 'No profile data in file' with the **c** option if the **z** option was not enabled to *pi;* 'Not a Pascal system core file' if the core is not from a *px* execution; 'Program and count data do not correspond' if the program was changed after compilation, before profiling; or if the wrong program is specified.

AUTHOR

William Joy

BUGS

Does not place multiple statements per line.

NAME

pxref − Pascal cross-reference program

SYNOPSIS

pxref [−] name

DESCRIPTION

Pxref makes a line numbered listing and a cross-reference of identifier usage for the program in *name*. The optional '−' argument suppresses the listing. The keywords **goto** and **label** are treated as identifiers for the purpose of the cross-reference. **Include** directives are not processed, but cause the placement of an entry indexed by '#include' in the cross-reference.

SEE ALSO

Berkeley Pascal User's Manual

AUTHOR

Niklaus Wirth

BUGS

Identifiers are trimmed to 10 characters.

NAME

quota − display disc usage and limits

SYNOPSIS

quota [**−qv**] [user]

DESCRIPTION

Quota displays users' disc usage and limits. Only the super-user may use the optional *user* argument to view the limits of users other than himself.

The **−q** flag prints a more terse message, containing only information on file systems where usage is over quota.

If a **−v** flag is supplied, *quota* will also display user's quotas on file systems where no storage is allocated.

Quota reports only on file systems which have disc quotas. If *quota* exits with a non-zero status, one or more file systems are over quota.

SEE ALSO

quota(2), quotaon(8)

NAME

ranlib — convert archives to random libraries

SYNOPSIS

ranlib archive ...

DESCRIPTION

Ranlib converts each *archive* to a form which the loader can load more rapidly. *Ranlib* does this by adding a table of contents called **_.SYMDEF** to the beginning of the archive. *Ranlib* uses *ar*(1) to reconstruct the archive, so that sufficient temporary file space must be available in the file system which contains the current directory.

SEE ALSO

ld(1), ar(1), lorder(1)

BUGS

Because generation of a library by *ar* and randomization of the library by *ranlib* are separate processes, phase errors are possible. The loader, *ld*, warns when the modification date of a library is more recent than the creation date of its dictionary; but this means that you get the warning even if you only copy the library.

NAME

ratfor − rational Fortran dialect

SYNOPSIS

ratfor [option ...] [filename ...]

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

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
```

and some syntactic sugar to make programs easier to read and write:

free form input:

multiple statements/line; automatic continuation

comments:

```
# this is a comment
```

translation of relationals:

>, >=, etc., become .GT., .GE., etc.

return (expression)

returns expression to caller from function

define: define name replacement

include:

include filename

Ratfor is best used with *f77*(1).

SEE ALSO

f77(1)

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

NAME

rcp − remote file copy

SYNOPSIS

rcp file1 file2
rcp [**−r**] file ... directory

DESCRIPTION

Rcp copies files between machines. Each *file* or *directory* argument is either a remote file name of the form "rhost:path", or a local file name (containing no ':' characters, or a '/' before any ':'s.)

If the **−r** is specified and any of the source files are directories, *rcp* copies each subtree rooted at that name; in this case the destination must be a directory.

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 (using \, ", or ') 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 *rsh*(1C).

Rcp handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form "rhost.rname" to use *rname* rather than the current user name on the remote host.

SEE ALSO

ftp(1C), rsh(1C), rlogin(1C)

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.

NAME

rcsintro - introduction to RCS commands

DESCRIPTION

The Revision Control System (RCS) manages multiple revisions of text files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, for example programs, documentation, graphics, papers, form letters, etc.

The basic user interface is extremely simple. The novice only needs to learn two commands: *ci* and *co*. *Ci*, short for "checkin", deposits the contents of a text file into an archival file called an RCS file. An RCS file contains all revisions of a particular text file. *Co*, short for "checkout", retrieves revisions from an RCS file.

Functions of RCS

- Storage and retrieval of multiple revisions of text. RCS saves all old revisions in a space efficient way. Changes no longer destroy the original, because the previous revisions remain accessible. Revisions can be retrieved according to ranges of revision numbers, symbolic names, dates, authors, and states.
- Maintenance of a complete history of changes. RCS logs all changes automatically. Besides the text of each revision, RCS stores the author, the date and time of checkin, and a log message summarizing the change. The logging makes it easy to find out what happened to a module, without having to compare source listings or having to track down colleagues.
- Resolution of access conflicts. When two or more programmers wish to modify the same revision, RCS alerts the programmers and prevents one modification from corrupting the other.
- Maintenance of a tree of Revisions. RCS can maintain separate lines of development for each module. It stores a tree structure that represents the ancestral relationships among revisions.
- Merging of revisions and resolution of conflicts. Two separate lines of development of a module can be coalesced by merging. If the revisions to be merged affect the same sections of code, RCS alerts the user about the overlapping changes.
- Release and configuration control. Revisions can be assigned symbolic names and marked as released, stable, experimental, etc. With these facilities, configurations of modules can be described simply and directly.
- Automatic identification of each revision with name, revision number, creation time, author, etc. The identification is like a stamp that can be embedded at an appropriate place in the text of a revision. The identification makes it simple to determine which revisions of which modules make up a given configuration.
- Minimization of secondary storage. RCS needs little extra space for the revisions (only the differences). If intermediate revisions are deleted, the corresponding deltas are compressed accordingly.

Getting Started with RCS

Suppose you have a file f.c that you wish to put under control of RCS. Invoke the checkin command

```
ci f.c
```

This command creates the RCS file f.c,v, stores f.c into it as revision 1.1, and deletes f.c. It also asks you for a description. The description should be a synopsis of the contents of the file. All later checkin commands will ask you for a log entry, which should summarize the changes that you made.

Files ending in ,v are called RCS files ('v' stands for 'versions'), the others are called working files. To get back the working file f.c in the previous example, use the checkout command

co f.c

This command extracts the latest revision from f.c,v and writes it into f.c. You can now edit f.c and check it back in by invoking

ci f.c

Ci increments the revision number properly. If *ci* complains with the message

ci error: no lock set by <your login>

then your system administrator has decided to create all RCS files with the locking attribute set to 'strict'. In this case, you should have locked the revision during the previous checkout. Your last checkout should have been

co -l f.c

Of course, it is too late now to do the checkout with locking, because you probably modified f.c already, and a second checkout would overwrite your modifications. Instead, invoke

rcs -l f.c

This command will lock the latest revision for you, unless somebody else got ahead of you already. In this case, you'll have to negotiate with that person.

Locking assures that you, and only you, can check in the next update, and avoids nasty problems if several people work on the same file. Even if a revision is locked, it can still be checked out for reading, compiling, etc. All that locking prevents is a CHECKIN by anybody but the locker.

If your RCS file is private, i.e., if you are the only person who is going to deposit revisions into it, strict locking is not needed and you can turn it off. If strict locking is turned off, the owner off the RCS file need not have a lock for checkin; all others still do. Turning strict locking off and on is done with the commands

rcs -U f.c and rcs -L f.c

If you don't want to clutter your working directory with RCS files, create a subdirectory called RCS in your working directory, and move all your RCS files there. RCS commands will look first into that directory to find needed files. All the commands discussed above will still work, without any modification. (Actually, pairs of RCS and working files can be specified in 3 ways: (a) both are given, (b) only the working file is given, (c) only the RCS file is given. Both RCS and working files may have arbitrary path prefixes; RCS commands pair them up intelligently).

To avoid the deletion of the working file during checkin (in case you want to continue editing), invoke

ci -l f.c or ci -u f.c

These commands check in f.c as usual, but perform an implicit checkout. The first form also locks the checked in revision, the second one doesn't. Thus, these options save you one checkout operation. The first form is useful if locking is strict, the second one if not strict. Both update the identification markers in your working file (see below).

You can give *ci* the number you want assigned to a checked in revision. Assume all your revisions were numbered 1.1, 1.2, 1.3, etc., and you would like to start release 2. The command

ci -r2 f.c or ci -r2.1 f.c

assigns the number 2.1 to the new revision. From then on, *ci* will number the subsequent revisions with 2.2, 2.3, etc. The corresponding *co* commands

co -r2 f.c and co -r2.1 f.c

retrieve the latest revision numbered 2.x and the revision 2.1, respectively. *Co* without a revision number selects the latest revision on the "trunk", i.e., the highest revision with a number consisting of 2 fields. Numbers with more than 2 fields are needed for branches. For example, to start a branch at revision 1.3, invoke

ci -r1.3.1 f.c

This command starts a branch numbered 1 at revision 1.3, and assigns the number 1.3.1.1 to the new revision. For more information about branches, see *rcsfile*(5).

Automatic Identification

RCS can put special strings for identification into your source and object code. To obtain such identification, place the marker

$Header$

into your text, for instance inside a comment. RCS will replace this marker with a string of the form

$Header: filename revision_number date time author state $

With such a marker on the first page of each module, you can always see with which revision you are working. RCS keeps the markers up to date automatically. To propagate the markers into your object code, simply put them into literal character strings. In C, this is done as follows:

static char rcsid[] = "$Header$";

The command *ident* extracts such markers from any file, even object code and dumps. Thus, *ident* lets you find out which revisions of which modules were used in a given program.

You may also find it useful to put the marker Log into your text, inside a comment. This marker accumulates the log messages that are requested during checkin. Thus, you can maintain the complete history of your file directly inside it. There are several additional identification markers; see *co*(1) for details.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.0 ; Release Date: 83/05/11 .

SEE ALSO

ci(1), co(1), ident(1), merge(1), rcs(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(5).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

NAME

rcs − change RCS file attributes

SYNOPSIS

rcs [options] file ...

DESCRIPTION

Rcs creates new RCS files or changes attributes of existing ones. An RCS file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For *rcs* to work, the caller's login name must be on the access list, except if the access list is empty, the caller is the owner of the file or the superuser, or the **-i** option is present.

Files ending in ',v' are RCS files, all others are working files. If a working file is given, *rcs* tries to find the corresponding RCS file first in directory ./RCS and then in the current directory, as explained in *co* (1).

−i creates and initializes a new RCS file, but does not deposit any revision. If the RCS file has no path prefix, *rcs* tries to place it first into the subdirectory ./RCS, and then into the current directory. If the RCS file already exists, an error message is printed.

−a*logins* appends the login names appearing in the comma-separated list *logins* to the access list of the RCS file.

−A*oldfile* appends the access list of *oldfile* to the access list of the RCS file.

−e[*logins*] erases the login names appearing in the comma-separated list *logins* from the access list of the RCS file. If *logins* is omitted, the entire access list is erased.

−c*string* sets the comment leader to *string*. The comment leader is printed before every log message line generated by the keyword Log during checkout (see *co*). This is useful for programming languages without multi-line comments. During *rcs -i* or initial *ci*, the comment leader is guessed from the suffix of the working file.

−l[*rev*] locks the revision with number *rev*. If a branch is given, the latest revision on that branch is locked. If *rev* is omitted, the latest revision on the trunk is locked. Locking prevents overlapping changes. A lock is removed with *ci* or *rcs -u* (see below).

−u[*rev*] unlocks the revision with number *rev*. If a branch is given, the latest revision on that branch is unlocked. If *rev* is omitted, the latest lock held by the caller is removed. Normally, only the locker of a revision may unlock it. Somebody else unlocking a revision breaks the lock. This causes a mail message to be sent to the original locker. The message contains a commentary solicited from the breaker. The commentary is terminated with a line containing a single '.' or control-D.

−L sets locking to *strict*. Strict locking means that the owner of an RCS file is not exempt from locking for checkin. This option should be used for files that are shared.

−U sets locking to non-strict. Non-strict locking means that the owner of a file need not lock a revision for checkin. This option should NOT be used for files that are shared. The default (**-L** or **-U**) is determined by your system administrator.

−n*name*[:*rev*]

associates the symbolic name *name* with the branch or revision *rev*. *Rcs* prints an error message if *name* is already associated with another number. If *rev* is omitted, the symbolic name is deleted.

−N*name*[:*rev*]

same as **-n**, except that it overrides a previous assignment of *name*.

-o*range* deletes ("outdates") the revisions given by *range*. A range consisting of a single revision number means that revision. A range consisting of a branch number means the latest revision on that branch. A range of the form *rev1*-*rev2* means revisions *rev1* to *rev2* on the same branch, -*rev* means from the beginning of the branch containing *rev* up to and including *rev*, and *rev*- means from revision *rev* to the end of the branch containing *rev*. None of the outdated revisions may have branches or locks.

-q quiet mode; diagnostics are not printed.

-s*state*[:*rev*]
sets the state attribute of the revision *rev* to *state*. If *rev* is omitted, the latest revision on the trunk is assumed; If *rev* is a branch number, the latest revision on that branch is assumed. Any identifier is acceptable for *state*. A useful set of states is *Exp* (for experimental), *Stab* (for stable), and *Rel* (for released). By default, *ci* sets the state of a revision to *Exp*.

-t[*txtfile*] writes descriptive text into the RCS file (deletes the existing text). If *txtfile* is omitted, *rcs* prompts the user for text supplied from the std. input, terminated with a line containing a single '.' or control-D. Otherwise, the descriptive text is copied from the file *txtfile*. If the **-i** option is present, descriptive text is requested even if **-t** is not given. The prompt is suppressed if the std. input is not a terminal.

DIAGNOSTICS

The RCS file name and the revisions outdated are written to the diagnostic output. The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 otherwise.

FILES

The caller of the command must have read/write permission for the directory containing the RCS file and read permission for the RCS file itself. *Rcs* creates a semaphore file in the same directory as the RCS file to prevent simultaneous update. For changes, *rcs* always creates a new file. On successful completion, *rcs* deletes the old one and renames the new one. This strategy makes links to RCS files useless.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.1 ; Release Date: 83/04/04 .

SEE ALSO

co (1), ci (1), ident(1), rcsdiff (1), rcsintro (1), rcsmerge (1), rlog (1), rcsfile (5), sccstorcs (8).

Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

BUGS

NAME

rcsdiff − compare RCS revisions

SYNOPSIS

rcsdiff [**-b**] [**-cefhn**] [**-r***rev1*] [**-r***rev2*] file ...

DESCRIPTION

Rcsdiff runs *diff* (1) to compare two revisions of each RCS file given. A file name ending in ',v' is an RCS file name, otherwise a working file name. *Rcsdiff* derives the working file name from the RCS file name and vice versa, as explained in *co* (1). Pairs consisting of both an RCS and a working file name may also be specified.

The options **-b**, **-c**, **-e**, **-f**, and **-h** have the same effect as described in *diff* (1); option **-n** generates an edit script of the format used by RCS.

If both *rev1* and *rev2* are omitted, *rcsdiff* compares the latest revision on the trunk with the contents of the corresponding working file. This is useful for determining what you changed since the last checkin.

If *rev1* is given, but *rev2* is omitted, *rcsdiff* compares revision *rev1* of the RCS file with the contents of the corresponding working file.

If both *rev1* and *rev2* are given, *rcsdiff* compares revisions *rev1* and *rev2* of the RCS file.

Both *rev1* and *rev2* may be given numerically or symbolically.

EXAMPLES

The command

```
rcsdiff f.c
```

runs *diff* on the latest trunk revision of RCS file f.c,v and the contents of working file f.c.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.0 ; Release Date: 83/01/15 .

SEE ALSO

ci (1), co (1), diff (1), ident (1), rcs (1), rcsintro (1), rcsmerge (1), rlog (1), rcsfile (5).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

BUGS

NAME

rcsmerge − merge RCS revisions

SYNOPSIS

rcsmerge -r*rev1* [**-r***rev2*] [**-p**] file

DESCRIPTION

Rcsmerge incorporates the changes between *rev1* and *rev2* of an RCS file into the corresponding working file. If **-p** is given, the result is printed on the std. output, otherwise the result overwrites the working file.

A file name ending in ',v' is an RCS file name, otherwise a working file name. *Merge* derives the working file name from the RCS file name and vice versa, as explained in *co* (1). A pair consisting of both an RCS and a working file name may also be specified.

Rev1 may not be omitted. If *rev2* is omitted, the latest revision on the trunk is assumed. Both *rev1* and *rev2* may be given numerically or symbolically.

Rcsmerge prints a warning if there are overlaps, and delimits the overlapping regions as explained in *co -j*. The command is useful for incorporating changes into a checked-out revision.

EXAMPLES

Suppose you have released revision 2.8 of f.c. Assume furthermore that you just completed revision 3.4, when you receive updates to release 2.8 from someone else. To combine the updates to 2.8 and your changes between 2.8 and 3.4, put the updates to 2.8 into file f.c and execute

```
rcsmerge -p -r2.8 -r3.4 f.c >f.merged.c
```

Then examine f.merged.c. Alternatively, if you want to save the updates to 2.8 in the RCS file, check them in as revision 2.8.1.1 and execute *co -j*:

```
ci -r2.8.1.1 f.c
co -r3.4 -j2.8:2.8.1.1 f.c
```

As another example, the following command undoes the changes between revision 2.4 and 2.8 in your currently checked out revision in f.c.

```
rcsmerge -r2.8 -r2.4 f.c
```

Note the order of the arguments, and that f.c will be overwritten.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.0 ; Release Date: 83/01/15 .

SEE ALSO

ci (1), co (1), merge (1), ident (1), rcs (1), rcsdiff (1), rlog (1), rcsfile (5).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

BUGS

Rcsmerge does not work for files that contain lines with a single '.'.

NAME

rdist − remote file distribution program

SYNOPSIS

rdist [−nqblRvwy] [−f distfile] [−d var=value] [name ...]

rdist [−nqblRvwy] -c name ... host[.login][:dest]

DESCRIPTION

Rdist is a program to maintain identical copies of files over multiple hosts. It preserves the owner, group, mode, and mtime of files if possible and can update programs that are executing. *Rdist* reads commands from *distfile* to direct the updating of files and/or directories. If *distfile* is '−', the standard input is used. If no −f option is present, the file 'distfile' is used for input. If no names are specified on the command line, *rdist* will update all of the files and directories listed in *distfile*. Otherwise, only the listed files will be updated.

The −c option forces *rdist* to interpret the remaining arguments as a small *distfile*. The equivalent distfile is as follows.

(*name* ...) -> *host*[*.login*]
install *dest* ;

Other options:

−d Define *var* to have *value*. The −d option is used to define or override variable definitions in the *distfile*. *Value* can be the empty string, one name, or a list of names surrounded by parentheses and separated by tabs and/or spaces.

−l Follow symbolic links on the source host as if they were ordinary files or directories.

−n Print the commands without executing them. This option is useful for debugging *distfile*.

−q Quiet mode. Files that are being modified are normally printed on standard output. The −q option suppresses this.

−R Remove extraneous files. If a directory is being updated, any files that exist on the remote host that do not exist in the master directory are removed. This is useful for maintaining truely identical copies of directories.

−v Verify that the files are up to date on all the hosts. Any files that are out of date will be displayed but no files will be changed nor any mail sent.

−w Whole mode. The whole file name is appended to the destination directory name. Normally, only the last component of a name is used when renaming files. This will preserve the directory structure of the files being copied instead of flattening the directory structure.

−y Younger mode. Files are normally updated if their *mtime* and *size* (see *stat*(2)) disagree. The −y option causes *rdist* to only update files that are younger than the master copy. This can be used to prevent newer copies on other hosts from being replaced. A warning message is printed for files which are newer than the master copy.

−b Binary comparison. Perform a binary comparison and update files if they differ rather than comparing dates and sizes.

Distfile contains a sequence of entries that specify the files to be copied, the destination hosts, and what operations to perform to do the updating. Each entry has one of the following formats.

<variable name> '=' <name list>
<source list> '−>' <destination list> <command list>
<source list> '::' <time_stamp file> <command list>

The first format is used for defining variables. The second format is used for distributing files to other hosts. The third format is used for making lists of files that have been changed since some given date. The *source list* specifies a list of files and/or directories on the local host which are to be used as the master copy for distribution. The *destination list* is the list of hosts these files are to be copied to. Each file in the source list is added to a list of changes if the file is out of date on the host being updated (second format) or the file is newer than the time stamp file (third format).

Newlines, tabs, and blanks are only used as separators and are otherwise ignored. Comments begin with '#' and end with a newline.

The source and destination lists have the following format:

<name>

or

'(' <zero or more names separated by white-space> ')'

The shell meta-characters '[', ']', '{', '}', '*', and '?' are recognized and expanded (on the local host only) in the same way as *csh*(1). The '~' character is also expanded in the same way as *csh* but is expanded separately on the local and destination hosts. When the −w option is used with a file name that begins with '~', everything except the home directory is appended to the destination name.

The command list consists of zero or more commands of the following format.

'install' <options> opt_dest_name ';'
'notify' <name list> ';'
'except' <name list> ';'
'special' <name list> string ';'

The *install* command is used to copy out of date files and/or directories. Each source file is copied to each host in the destination list. Directories are recursively copied in the same way. *Opt_dest_name* is an optional parameter to rename files. If no *install* command appears in the command list or the destination name is not specified, the source file name is used. Directories in the path name will be created if they do not exist on the remote host. The *options* are '−R', '−v', '−w', '−y', and '−b' and have the same semantics as options on the command line except they only apply to the files in the source list. The login name used on the destination host is the same as the local host unless the destination name is of the format "host.login".

The *notify* command is used to mail the list of files updated (and any errors that may have occured) to the listed names. If no '@' appears in the name, the destination host is appended to the name (e.g., name1@host, name2@host, ...).

The *except* command is used to update all of the files in the source list **except** for the files listed in *name list*. This is mostly used to copy everything in a directory except certain files.

The *special* command is used to specify shell commands that are to be executed on the remote host after the file in *name list* is updated or installed. *String* starts and ends with "" and can cross multiple lines in *distfile*. Multiple commands to the shell should be separated by ';'. The *special* command can be used to create links, rebuild private databases, etc. after a program has been updated.

The following is a small example.

```
HOSTS = ( matisse arpa.root )

FILES = ( /bin /lib /usr/bin /usr/games
        /usr/include/{*.h,{stand,sys,vax*,pascal,machine}/*.h}
```

```
        /usr/lib /usr/man/man? /usr/ucb /usr/local/rdist )

EXLIB = ( Mail.rc aliases aliases.dir aliases.pag crontab dshrc
        sendmail.cf sendmail.fc sendmail.hf sendmail.st uucp vfont )

${FILES} -> ${HOSTS}
        install -r ;
        except /usr/lib/${EXLIB} ;
        except /usr/games/lib ;
        except /usr/ucb/f ;
        special /usr/ucb/finger "rm /usr/ucb/f; ln /usr/ucb/finger /usr/ucb/f" ;

IMAGEN = (ips dviimp catdvi)
/usr/local/${IMAGEN} -> arpa
        install /usr/local/lib ;
        notify ralph ;

${FILES} :: stamp.cory
        notify root@cory ;
```

FILES

distfile	input command file
/tmp/rdist*	temporary file for update lists

SEE ALSO

csh(1), stat(2)

AUTHOR

Ralph Campbell

BUGS

Source files must reside on the local host where rdist is executed.

The names used to update specific files from *distfile* must match the expanded name (i.e. rdist "${FILES}" will not work).

NAME

rlog − print log messages and other information about RCS files

SYNOPSIS

rlog [options] file ...

DESCRIPTION

Rlog prints information about RCS files. Files ending in ',v' are RCS files, all others are working files. If a working file is given, *rlog* tries to find the corresponding RCS file first in directory ./RCS and then in the current directory, as explained in *co* (1).

Rlog prints the following information for each RCS file: RCS file name, working file name, head (i.e., the number of the latest revision on the trunk), access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for printing, and descriptive text. This is followed by entries for the selected revisions in reverse chronological order for each branch. For each revision, *rlog* prints revision number, author, date/time, state, number of lines added/deleted (with respect to the previous revision), locker of the revision (if any), and log message. Without options, *rlog* prints complete information. The options below restrict this output.

−L ignores RCS files that have no locks set; convenient in combination with **-R**, **-h**, or **-l**.

−R only prints the name of the RCS file; convenient for translating a working file name into an RCS file name.

−h prints only RCS file name, working file name, head, access list, locks, symbolic names, and suffix.

−t prints the same as **-h**, plus the descriptive text.

−d*dates* prints information about revisions with a checkin date/time in the ranges given by the semicolon-separated list of *dates*. A range of the form *d1*<*d2* or *d2*>*d1* selects the revisions that were deposited between *d1* and *d2*, (inclusive). A range of the form <*d* or *d*> selects all revisions dated *d* or earlier. A range of the form *d*< or >*d* selects all revisions dated *d* or later. A range of the form *d* selects the single, latest revision dated *d* or earlier. The date/time strings *d*, *d1*, and *d2* are in the free format explained in *co* (1). Quoting is normally necessary, especially for < and >. Note that the separator is a semicolon.

−l[*lockers*]
prints information about locked revisions. If the comma-separated list *lockers* of login names is given, only the revisions locked by the given login names are printed. If the list is omitted, all locked revisions are printed.

−r*revisions*
prints information about revisions given in the comma-separated list *revisions* of revisions and ranges. A range *rev1*−*rev2* means revisions *rev1* to *rev2* on the same branch, −*rev* means revisions from the beginning of the branch up to and including *rev*, and *rev*− means revisions starting with *rev* to the end of the branch containing *rev*. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range.

−s*states* prints information about revisions whose state attributes match one of the states given in the comma-separated list *states*.

−w[*logins*]
prints information about revisions checked in by users with login names appearing in the comma-separated list *logins*. If *logins* is omitted, the user's login is assumed.

NAME

readnews – read news articles

SYNOPSIS

readnews [**–a** *date*] [**–n** *newsgroups*] [**–t** *titles*] [**–lprxhfuM**] [**–c** [*mailer*]]

readnews –s

DESCRIPTION

readnews without argument prints unread articles. There are several interfaces available:

Flag	Interface
default	A *msgs*(1) like interface.
–M	An interface to *Mail*(1).
–c	A */bin/mail*(1)–like interface.

–c "*mailer*"
All selected articles written to a temporary file. Then the mailer is invoked. The name of the temporary file is referenced with a "%". Thus, "mail –f %" will invoke mail on a temporary file consisting of all selected messages.

–p All selected articles are sent to the standard output. No questions asked.

–l Only the titles output. The *.newsrc* file will not be updated.

The **–r** flag causes the articles to be printed in reverse order. The **–f** flag prevents any followup articles from being printed. The **–h** flag causes articles to be printed in a less verbose format, and is intended for terminals running at 300 baud. the **–u** flag causes the **.newsrc** file to be updated every 5 minutes, in case of an unreliable system. (Note that if the newsrc file is updated, the **x** command will not restore it to its original contents.)

The following flags determine the selection of articles.

–n *newsgroups*
Select all articles that belong to *newsgroups*.

–t *titles* Select all articles whose titles contain one of the strings specified by *titles*.

–a [*date*]
Select all articles that were posted past the given *date* (in *getdate*(3) format).

–x Ignore *.newsrc* file. That is, select articles that have already been read as well as new ones.

readnews maintains a *.newsrc* file in the user's home directory that specifies all news articles already read. It is updated at the end of each reading session in which the **–x** or **–l** options weren't specified. If the environment variable NEWSRC is present, it should be the path name of a file to be used in place of .newsrc.

If the user wishes, an options line may be placed in the *.newsrc* file. This line starts with the word **options** (left justified) followed by the list of standard options just as they would be typed on the command line. Such a list may include: the **–n** flag along with a newsgroup list; a favorite interface; and/or the

-r or **-t** flag. Continuation lines are specified by following lines beginning with a space or tab character. Similarly, options can be specified in the **NEWSOPTS** environment parameter. Where conflicts exist, option on the command line take precedence, followed by the *.newsrc* **options** line, and lastly the **NEWSOPTS** parameter.

readnews -s will print the newsgroup subscription list.

When the user uses the reply command of the *msgs*(1) or */bin/mail*(1) interfaces, the environment parameter **MAILER** will be used to determine which mailer to use. The default is usually /bin/mail.

If the user so desires, he may specify a specific paging progam for articles. The environment parameter **PAGER** should be set to the paging program. The name of the article is referenced with a '%', as in the **-c** option. If no '%' is present, the article will be piped to the program. Paging may be disabled by setting **PAGER** to a null value.

COMMANDS

This section lists the commands you can type to the msgs and /bin/mail interface prompts. The msgs interface will suggest some common commands in brackets. Just hitting return is the same as typing the first command. For example, "[ynq]" means that the commands "y" (yes), "n" (no), and "q" (quit) are common responses, and that "y" is the default.

Command Meaning

y Yes. Prints current article and goes on to next.

n No. Goes on to next article without printing current one. In the /bin/mail interface, this means "go on to the next article", which will have the same effect as "y" or just hitting return.

q Quit. The .newsrc file will be updated if -l or -x were not on the command line.

c Cancel the article. Only the author or the super user can do this.

r Reply. Reply to article's author via mail. You are placed in your EDITOR with a header specifying To, Subject, and References lines taken from the message. You may change or add headers, as appropriate. You add the text of the reply after the blank line, and then exit the editor. The resulting message is mailed to the author of the article.

rd Reply directly. You are placed in $MAILER ("mail" by default) in reply to the author. Type the text of the reply and then control-D.

f [*title*]
Submit a follow up article. Normally you should leave off the title, since the system will generate one for you. You will be placed in your EDITOR to compose the text of the followup.

fd Followup directly, without edited headers. This is like *f*, but the headers of the article are not included in the editor buffer.

N [*newsgroup*]
Go to the next newsgroup or named newsgroup.

s [*file*]
Save. The article is appended to the named file. The default is "Articles". If the first character of the file name is '|', the rest of the file name is taken as the name of a program, which is executed with the text of the

article as standard input. If the first character of the file name is '/', it is taken as a full path name of a file. If $NEWSBOX (in the environment) is set to a full path name, and the file contains no '/', the file is saved in $NEWSBOX. Otherwise, it is saved relative to $HOME.

Report the name and size of the newsgroup.

e Erase. Forget that this article was read.

h Print a more verbose header.

H Print a very verbose header, containing all known information about the article.

U Unsubscribe from this newsgroup. Also goes on to the next newsgroup.

d Read a digest. Breaks up a digest into separate articles and permits you to read and reply to each piece.

D Decrypt. Invokes a Caesar decoding program on the body of the message. This is used to decrypt rotated jokes posted to net.jokes. Such jokes are usually obscene or otherwise offensive to some groups of people, and so are rotated to avoid accidental decryption by people who would be offended. The title of the joke should indicate the nature of the problem, enabling people to decide whether to decrypt it or not.

Normally the Caesar program does a character frequency count on each line of the article separately, so that lines which are not rotated will be shown in plain text. This works well unless the line is short, in which case it sometimes gets the wrong rotation. An explicit *number* rotation (usually 13) may be given to force a particular shift.

v Print the current version of the news software.

! Shell escape.

number
Go to *number*.

+[*n*] Skip n articles. The articles skipped are recorded as "unread" and will be offered to you again the next time you read news.

– Go back to last article. This is a toggle, typing it twice returns you to the original article.

x Exit. Like quit except that .newsrc is not updated.

X *system*
Transmit article to the named system.

The commands c, f, fd, r, rd, e, h, H, and s can be followed by –'s to refer to the previous article. Thus, when replying to an article using the msgs interface, you should normally type "r–" (or "re-") since by the time you enter a command, you are being offerred the next article.

EXAMPLES

readnews
Read all unread articles using the *msgs*(1) interface. The *.newsrc* file is updated at the end of the session.

readnews –c "ed %" –l
Invoke the *ed*(1) text editor on a file containing the titles of all unread articles. The *.newsrc* file is **not** updated at the end of the session.

readnews -n all !fa.all -M -r
Read all unread articles except articles whose newsgroups begin with "fa." via *Mail*(1) in reverse order. The *.newsrc* file is updated at the end of the session.

readnews -p -n all -a last thursday
Print every unread article since last Thursday. The *.newsrc* file is updated at the end of the session.

readnews -p > /dev/null &
Discard all unread news. This is useful after returning from a long trip.

FILES

/usr/spool/news/*newsgroup/number*	News articles
/usr/lib/news/active	Active newsgroups and numbers of articles
/usr/lib/news/help	Help file for *msgs*(1) interface
~/.newsrc	Options and list of previously read articles

SEE ALSO

checknews(1), inews(1), sendnews(8), recnews(8), uurec(8), msgs(1), Mail(1), mail(1), news(5), newsrc(5)

AUTHORS

Matt Glickman
Mark Horton
Stephen Daniel
Tom R. Truscott

NAME

refer − find and insert literature references in documents

SYNOPSIS

refer [**−a**] [**−b**] [**−c**] [**−e**] [**−f***n*] [**−k***x*] [**−l***m,n*] [**−n**] [**−p** bib] [**−s***keys*] [**−B***l,m*] [**−P**] [**−S**] [file ...]

DESCRIPTION

Refer is a preprocessor for *nroff* or *troff*(1) that finds and formats references for footnotes or endnotes. It is also the base for a series of programs designed to index, search, sort, and print stand-alone bibliographies, or other data entered in the appropriate form.

Given an incomplete citation with sufficiently precise keywords, *refer* will search a bibliographic database for references containing these keywords anywhere in the title, author, journal, etc. The input file (or standard input) is copied to standard output, except for lines between .[and .] delimiters, which are assumed to contain keywords, and are replaced by information from the bibliographic database. The user may also search different databases, override particular fields, or add new fields. The reference data, from whatever source, are assigned to a set of *troff* strings. Macro packages such as *ms*(7) print the finished reference text from these strings. By default references are flagged by footnote numbers.

The following options are available:

−a*r* Reverse the first *n* author names (Jones, J. A. instead of J. A. Jones). If *n* is omitted all author names are reversed.

−b Bare mode: do not put any flags in text (neither numbers nor labels).

−c*keys*
Capitalize (with CAPS SMALL CAPS) the fields whose key-letters are in *keys*.

−e Instead of leaving the references where encountered, accumulate them until a sequence of the form

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to same source.

−f*n* Set the footnote number to *n* instead of the default of 1 (one). With labels rather than numbers, this flag is a no-op.

−k*x* Instead of numbering references, use labels as specified in a reference data line beginning *%x;* by default *x* is **L**.

−l*m,n*
Instead of numbering references, use labels made from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either *m* or *n* is omitted the entire name or date respectively is used.

−n Do not search the default file /usr/dict/papers/Ind. If there is a REFER environment variable, the specified file will be searched instead of the default file; in this case the **−n** flag has no effect.

−p *bib*
Take the next argument *bib* as a file of references to be searched. The default file is searched last.

−s*keys*
Sort references by fields whose key-letters are in the *keys* string; permute reference

numbers in text accordingly. Implies **−e**. The key-letters in *keys* may be followed by a number to indicate how many such fields are used, with **+** taken as a very large number. The default is **AD** which sorts on the senior author and then date; to sort, for example, on all authors and then title use **-sA+T**.

−B*l.m*
Bibliography mode. Take a file composed of records separated by blank lines, and turn them into *troff* input. Label *l* will be turned into the macro *.m* with *l* defaulting to **%X** and *.m* defaulting to **.AP** (annotation paragraph).

−P Place punctuation marks .,:;?! after the reference signal, rather than before. (Periods and commas used to be done with strings.)

−S Produce references in the Natural or Social Science format.

To use your own references, put them in the format described below. They can be searched more rapidly by running *indxbib*(1) on them before using *refer;* failure to index results in a linear search. When *refer* is used with the *eqn, neqn* or *tbl* preprocessors *refer* should be first, to minimize the volume of data passed through pipes.

The *refer* preprocessor and associated programs expect input from a file of references composed of records separated by blank lines. A record is a set of lines (fields), each containing one kind of information. Fields start on a line beginning with a "%", followed by a key-letter, then a blank, and finally the contents of the field, and continue until the next line starting with "%". The output ordering and formatting of fields is controlled by the macros specified for *nroff/troff* (for footnotes and endnotes) or *roffbib* (for stand-alone bibliographies). For a list of the most common key-letters and their corresponding fields, see *addbib*(1). An example of a *refer* entry is given below.

EXAMPLE

```
%A      M. E. Lesk
%T      Some Applications of Inverted Indexes on the UNIX System
%B      UNIX Programmer's Manual
%V      2b
%I      Bell Laboratories
%C      Murray Hill, NJ
%D      1978
```

FILES

/usr/dict/papers	directory of default publication lists
/usr/lib/refer	directory of companion programs

SEE ALSO

addbib(1), sortbib(1), roffbib(1), indxbib(1), lookbib(1)

AUTHOR

Mike Lesk

BUGS

Blank spaces at the end of lines in bibliography fields will cause the records to sort and reverse incorrectly. Sorting large numbers of references causes a core dump.

NAME

refile -- file message(s) in (an)other folder(s)

SYNOPSIS

refile [−src +folder] [msgs] [−link] [−preserve] +folder ... [−nolink] [−nopreserve] [−file file] [−nofile] [−help]

DESCRIPTION

Refile moves (*mv*(1)) or links (*ln*(1)) messages from a source folder into one or more destination folders. If you think of a message as a sheet of paper, this operation is not unlike filing the sheet of paper (or copies) in file cabinet folders. When a message is refiled, it is linked into the destination folder(s) if possible, and is copied otherwise. As long as the destination folders are all on the same file system, multiple filing causes little storage overhead. This facility provides a good way to cross-file or multiply-index messages. For example, if a message is received from Jones about the ARPA Map Project, the command

refile cur +jones +Map

would allow the message to be found in either of the two folders 'jones' or 'Map'.

The option '−file file' directs *refile* to use the specified file as the source message to be filed, rather than a message from a folder.

If a destination folder doesn't exist, *refile* will ask if you want to create one. A negative response will abort the refile operation.

'−link' preserves the source folder copy of the message (i.e., it does a *ln*(1) rather than a *mv*(1)), whereas, '−nolink' deletes the "refiled" messages from the source folder. Normally, when a message is refiled, it is assigned the next highest number available in each of the destination folders. Use of the '−preserve' switch will override this message "renaming", but name conflicts may occur, so use this switch cautiously. (See *pick* for more details on message numbering.)

If '−link' is not specified (or '−nolink' is specified), the refiled messages will be removed (*unlink*(2)) from the source folder.

If '−src +folder' is given, it will become the current folder for future MH commands. If neither '−link' nor 'all' are specified, the current message in the source folder will be set to the last message specified; otherwise, the current message won't be changed.

FILES

$HOME/mh_profile The user profile

PROFILE COMPONENTS

Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder
Folder−Protect:	To set mode when creating a new folder

DEFAULTS

'−src +folder' defaults to the current folder
'msgs' defaults to cur
'−nolink'
'−nopreserve'
'−nofile'

CONTEXT

If '−src +folder' is given, it will become the current folder for future MH commands. If neither '−link' nor 'all' are specified, the current message in the source folder will be set to the last message specified; otherwise, the current message won't be changed.

NAME

remote – Remote command execution

SYNOPSIS

host **[-d]** *command [args...]*

DESCRIPTION

Remote is a program used to execute commands on another host in a PUP inter-network. It uses the PUP ftp protocol, so the commands can only be done on a host which is running an *ftp server* program, such as the Unix time-sharing systems. A special escape mechanism is used, so not even all ftp servers are guaranteed to work. You must have a valid user name and password stored in your home directory with the *netalias(1)* feature or the *pupftp(1)* program.

The *host* is specified as the "zeroth" argument, which means valid hosts will be linked to the remote program. If the "zeroth" argument is **remote**, then the first argument is the host name, and the remote command is the second through the last. For example, to finger the user named smith on the machine named Shasta, use the command

shasta finger smith

The standard output of the command is sent to the currently open standard output file, which may be redirected or piped in the usual manner. The **-d** option will print out some debugging information, like the ftp property lists being sent and recieved.

SEE ALSO

pupftp(1), telnet(1), netalias(1), ftpser(8)

AUTHOR

Bill Nowicki

BUGS

There should be a way of supplying input to a command, and executing a subset of commands (like "finger") on hosts on which the user does not have an account. This should be invoked automagically by a command of the form "finger user@host".

NAME

repl – reply to a message

SYNOPSIS

repl [+folder] [msg] [–editor editor] [–inplace] [–annotate] [–help] [–noinplace] [–noannotate]

DESCRIPTION

Repl aids a user in producing a reply to an existing message. In its simplest form (with no arguments), it will set up a message-form skeleton in reply to the current message in the current folder, invoke the editor, and send the composed message if so directed. The composed message is constructed as follows:

To: <Reply-To> or <From>
cc: <cc>, <To>
Subject: Re: <Subject>
In-reply-to: Your message of <Date>
<Message-Id>

where field names enclosed in angle brackets (< >) indicate the contents of the named field from the message to which the reply is being made. Once the skeleton is constructed, an editor is invoked (as in *comp*, *dist*, and *forw*). While in the editor, the message being replied to is available through a link named "@". In NED, this means the replied-to message may be "used" with "use @", or put in a window by "window @".

As in *comp*, *dist*, and *forw*, the user will be queried before the message is sent. If '–annotate' is specified, the replied-to message will be annotated with the single line

Replied: Date.

The command "comp –use" may be used to pick up interrupted editing, as in *dist* and *forw*; the '–inplace' switch annotates the message in place, so that all folders with links to it will see the annotation.

FILES

$HOME/mh_profile	The user profile
<mh-dir>/draft	The constructed message file
/usr/bin/send	To send the composed message

PROFILE COMPONENTS

Path:	To determine the user's MH directory
Editor:	To override the use of /bin/ned as the default editor
Current-Folder:	To find the default current folder

DEFAULTS

'+folder' defaults to current
'msgs' defaults to cur
'–editor' defaults to /bin/ned
'–noannotate'
'–noinplace'

CONTEXT

If a '+folder' is specified, it will become the current folder, and the current message will be set to the replied-to message.

NAME

reset — reset the teletype bits to a sensible state

SYNOPSIS

reset

DESCRIPTION

Reset sets the terminal to cooked mode, turns off cbreak and raw modes, turns on nl, and restores special characters that are undefined to their default values.

This is most useful after a program dies leaving a terminal in a funny state; you have to type "<LF>reset<LF>" to get it to work then to the shell, as <CR> often doesn't work; often none of this will echo.

It is a good idea to follow *reset* with *tset*(1)

SEE ALSO

stty(1), tset(1)

BUGS

Doesn't set tabs properly; it can't intuit personal choices for interrupt and line kill characters, so it leaves these set to the local system standards.

NAME

rev − reverse lines of a file

SYNOPSIS

rev [file] ...

DESCRIPTION

Rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

NAME

rev68 – reverse byte order in 68000 .b and .68 (b.out) files

SYNOPSIS

rev68 infile [outfile ...]

DESCRIPTION

Rev68 translates .b and .68 (b.out) files into a form readable by the 68000. This entails reversing the byte order within short and long words, and rearranging structs to agree with the c68 interpretation of structs.

The transformations that take place are: the 8 longs in the header (including the magic number) are each byte reversed; the text and data segments are unchanged; the symbol segment is modified so that the struct defining the type and value of the symbol fits in 6 bytes instead of 8 (to agree with the c68 interpretation of struct sym in /usr/sun/ld68/b.out.h); each symbol is padded with an extra 0 if necessary to force word alignment; and the relocation commands are repacked to agree with the c68 interpretation of struct reloc in /usr/sun/ld68/b.out.h.

The output is written to outfile, which if not specified defaults to r.out. The output will be smaller than the input unless there are no symbols or relocation commands.

The intent is that r.out files not be the input to any Unix program, but that they be sent directly to the 68000 as a byte stream.

AUTHOR

V.R. Pratt

NAME

rl68 -- print relocation commands in a .b file for the 68000

SYNOPSIS

rl68 infile

DESCRIPTION

Rl68 prints the relocation commands in a 68000 .b file. The format for each command is:

AREA SEGMENT SIZE Displacement Symbol

The AREA is one of T or D according to whether the command acts on Text or Data. The SEGMENT is one of T, D, B, or E, according to whether the object linked to is in Text, Data, or Bss, or is External. The SIZE is one of B, W, or L, according to whether the object linked to is of size Byte, Word, or Long. The Displacement indicates where in the AREA the command is to be applied. The Symbol is the symbolic name of the object linked to if any.

The correspondence with struct reloc (defined in /usr/sun/ld68/b.out.h) is as follows. SEGMENT is rsegment and SIZE is rsize. Displacement is rdisp. Symbol is determined from rsymbol, a short which is the symbol id. AREA is determined by the area in which the relocation command is found, there being two such areas.

RELATED

nm68, size68, pr68, lorder68

AUTHOR

C.J. Terman

Rlog prints the intersection of the revisions selected with the options **-d**, **-l**, **-s**, **-w**, intersected with the union of the revisions selected by **-b** and **-r**.

EXAMPLES

```
rlog -L -R RCS/*,v
rlog -L -h RCS/*,v
rlog -L -l RCS/*,v
rlog RCS/*,v
```

The first command prints the names of all RCS files in the subdirectory 'RCS' which have locks. The second command prints the headers of those files, and the third prints the headers plus the log messages of the locked revisions. The last command prints complete information.

DIAGNOSTICS

The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 otherwise.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.2 ; Release Date: 83/05/11 .

SEE ALSO

ci (1), co (1), ident(1), rcs (1), rcsdiff (1), rcsintro (1), rcsmerge (1), rcsfile (5), sccstorcs (8).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

BUGS

NAME

rlog − print log messages and other information about RCS files

SYNOPSIS

rlog [options] file ...

DESCRIPTION

Rlog prints information about RCS files. Files ending in ',v' are RCS files, all others are working files. If a working file is given, *rlog* tries to find the corresponding RCS file first in directory ./RCS and then in the current directory, as explained in *co* (1).

Rlog prints the following information for each RCS file: RCS file name, working file name, head (i.e., the number of the latest revision on the trunk), access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for printing, and descriptive text. This is followed by entries for the selected revisions in reverse chronological order for each branch. For each revision, *rlog* prints revision number, author, date/time, state, number of lines added/deleted (with respect to the previous revision), locker of the revision (if any), and log message. Without options, *rlog* prints complete information. The options below restrict this output.

−L ignores RCS files that have no locks set; convenient in combination with **-R**, **-h**, or **-l**.

−R only prints the name of the RCS file; convenient for translating a working file name into an RCS file name.

−h prints only RCS file name, working file name, head, access list, locks, symbolic names, and suffix.

−t prints the same as **-h**, plus the descriptive text.

−d*dates* prints information about revisions with a checkin date/time in the ranges given by the semicolon-separated list of *dates*. A range of the form *d1*<*d2* or *d2*>*d1* selects the revisions that were deposited between *d1* and *d2*, (inclusive). A range of the form <*d* or *d*> selects all revisions dated *d* or earlier. A range of the form *d*< or >*d* selects all revisions dated *d* or later. A range of the form *d* selects the single, latest revision dated *d* or earlier. The date/time strings *d*, *d1*, and *d2* are in the free format explained in *co* (1). Quoting is normally necessary, especially for < and >. Note that the separator is a semicolon.

−l[*lockers*]
prints information about locked revisions. If the comma-separated list *lockers* of login names is given, only the revisions locked by the given login names are printed. If the list is omitted, all locked revisions are printed.

−r*revisions*
prints information about revisions given in the comma-separated list *revisions* of revisions and ranges. A range *rev1*−*rev2* means revisions *rev1* to *rev2* on the same branch, −*rev* means revisions from the beginning of the branch up to and including *rev*, and *rev*− means revisions starting with *rev* to the end of the branch containing *rev*. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range.

−s*states* prints information about revisions whose state attributes match one of the states given in the comma-separated list *states*.

−w[*logins*]
prints information about revisions checked in by users with login names appearing in the comma-separated list *logins*. If *logins* is omitted, the user's login is assumed.

Rlog prints the intersection of the revisions selected with the options **-d**, **-l**, **-s**, **-w**, intersected with the union of the revisions selected by **-b** and **-r**.

EXAMPLES

```
rlog -L -R RCS/*,v
rlog -L -h RCS/*,v
rlog -L -l RCS/*,v
rlog RCS/*,v
```

The first command prints the names of all RCS files in the subdirectory 'RCS' which have locks. The second command prints the headers of those files, and the third prints the headers plus the log messages of the locked revisions. The last command prints complete information.

DIAGNOSTICS

The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 otherwise.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.2 ; Release Date: 83/05/11 .

SEE ALSO

ci (1), co (1), ident(1), rcs (1), rcsdiff (1), rcsintro (1), rcsmerge (1), rcsfile (5), sccstorcs (8).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

BUGS

NAME

rlogin − remote login

SYNOPSIS

rlogin rhost [−e *c*] [−l username]
rhost [−e*c*] [−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* which contains a list of *rhost*'s with which it shares account names. (The host names must be the standard names as described in *rsh*(1C).) When you *rlogin* as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file .rhosts in his login directory. Each line in this file should contain a *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 and may not be a symbolic link.

Your remote terminal type is the same as your local terminal type (as given in your environment TERM variable). All echoing takes place at the remote site, so that (except for delays) the rlogin is transparent. Flow control via ^S and ^Q and flushing of input and output on interrupts are handled properly. A line of the form "~." disconnects from the remote host, where "~" is the escape character. A different escape character may be specified by the −e option. There is no space separating this option flag and the argument character.

SEE ALSO

rsh(1C)

FILES

/usr/hosts/* for *rhost* version of the command

BUGS

More terminal characteristics should be propagated.

NAME

rm, rmdir − remove (unlink) files or directories

SYNOPSIS

rm [**−f**] [**−r**] [**−i**] [−] file ...

rmdir dir ...

DESCRIPTION

Rm 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, its permissions are printed and a line is read from the standard input. If that line begins with 'y' the file is deleted, otherwise the file remains. No questions are asked and no errors are reported when the **−f** (force) option is given.

If a designated file is a directory, an error comment is printed unless the optional argument **−r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **−i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **−r**, whether to examine each directory.

The null option − indicates that all the arguments following it are to be treated as file names. This allows the specification of file names starting with a minus.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

rm(1), unlink(2), rmdir(2)

NAME

rmail — handle remote mail received via uucp

SYNOPSIS

rmail user ...

DESCRIPTION

Rmail interprets incoming mail received via *uucp*(1C), collapsing "From" lines in the form generated by *binmail*(1) into a single line of the form "return-path!sender", and passing the processed mail on to *sendmail*(8).

Rmail is explicitly designed for use with *uucp* and *sendmail.*

SEE ALSO

binmail(1), uucp(1C), sendmail(8)

BUGS

Rmail should not reside in /bin.

NAME

rmdir, rm − remove (unlink) directories or files

SYNOPSIS

rmdir dir ...

rm [**−f**] [**−r**] [**−i**] [**−**] file ...

DESCRIPTION

Rmdir removes entries for the named directories, which must be empty.

Rm 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, its permissions are printed and a line is read from the standard input. If that line begins with 'y' the file is deleted, otherwise the file remains. No questions are asked and no errors are reported when the **−f** (force) option is given.

If a designated file is a directory, an error comment is printed unless the optional argument **−r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **−i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **−r**, whether to examine each directory.

The null option **−** indicates that all the arguments following it are to be treated as file names. This allows the specification of file names starting with a minus.

SEE ALSO

rm(1), unlink(2), rmdir(2)

NAME

rmf – remove folder

SYNOPSIS

rmf [+folder] [–help]

DESCRIPTION

Rmf removes all of the files (messages) within the specified (or default) folder, and then removes the directory (folder). If there are any files within the folder which are not a part of MH, they will *not* be removed, and an error will be produced. If the folder is given explicitly or the current folder is a subfolder (i.e., a selection list from *pick*), it will be removed without confirmation. If no argument is specified and the current folder is not a selection-list folder, the user will be asked for confirmation.

Rmf irreversibly deletes messages that don't have other links, so use it with caution.

If the folder being removed is a subfolder, the parent folder will become the new current folder, and *rmf* will produce a message telling the user this has happened. This provides an easy mechanism for selecting a set of messages, operating on the list, then removing the list and returning to the current folder from which the list was extracted. (See the example under *pick*.)

The files that *rmf* will delete are cur, any file beginning with a comma, and files with purely numeric names. All others will produce error messages.

Rmf of a read-only folder will delete the "cur–" entry from the profile without affecting the folder itself.

FILES

PROFILE COMPONENTS

$HOME/mh_profile	The user profile
Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder

DEFAULTS

'+folder' defaults to current, usually with confirmation

CONTEXT

Rmf will set the current folder to the parent folder if a subfolder is removed; or if the current folder is removed, it will make "inbox" current. Otherwise, it doesn't change the current folder or message.

NAME

rmm – remove messages

SYNOPSIS

rmm [+folder] [msgs] [–help]

DESCRIPTION

Rmm removes the specified messages by renaming the message files with preceding commas. (This is the Rand-UNIX backup file convention.)

The current message is not changed by *rmm*, so a *next* will advance to the next message in the folder as expected.

FILES

$HOME/mh_profile The user profile

PROFILE COMPONENTS

Path: To determine the user's MH directory
Current-Folder: To find the default current folder

DEFAULTS

'+folder' defaults to current
'msgs' defaults to cur

CONTEXT

If a folder is given, it will become current.

NAME

roffbib − run off bibliographic database

SYNOPSIS

roffbib [**−e**] [**−h**] [**−n**] [**−o**] [**−r**] [**−s**] [**−T***term*] [**−x**] [**−m** *mac*] [**−V**] [**−Q**] [file ...]

DESCRIPTION

Roffbib prints out all records in a bibliographic database, in bibliography format rather than as footnotes or endnotes. Generally it is used in conjunction with *sortbib*:

sortbib database | roffbib

Roffbib accepts most of the options understood by *nroff*(1), most importantly the **−T** flag to specify terminal type.

If abstracts or comments are entered following the %X field key, *roffbib* will format them into paragraphs for an annotated bibliography. Several %X fields may be given if several annotation paragraphs are desired. The **−x** flag will suppress the printing of these abstracts.

A user-defined set of macros may be specified after the **−m** option. There should be a space between the **−m** and the macro filename. This set of macros will replace the ones defined in /usr/lib/tmac/tmac.bib. The **−V** flag will send output to the Versatec; the **−Q** flag will queue output for the phototypesetter.

Four command-line registers control formatting style of the bibliography, much like the number registers of *ms*(7). The command-line argument **−rN1** will number the references starting at one (1). The flag **−rV2** will double space the bibliography, while **−rV1** will double space references but single space annotation paragraphs. The line length can be changed from the default 6.5 inches to 6 inches with the **−rL6i** argument, and the page offset can be set from the default of 0 to one inch by specifying **−rO1i** (capital O, not zero). Note: with the **−V** and **−Q** flags the default page offset is already one inch.

FILES

/usr/lib/tmac/tmac.bib file of macros used by *nroff/troff*

SEE ALSO

refer(1), addbib(1), sortbib(1), indxbib(1), lookbib(1)

AUTHORS

Greg Shenaut, Bill Tuthill

BUGS

Users have to rewrite macros to create customized formats.

NAME

rsh − remote shell

SYNOPSIS

rsh host [**−l** username] [**−n**] command
host [**−l** username] [**−n**] command

DESCRIPTION

Rsh connects to the specified *host,* and executes the specified *command. Rsh* 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; *rsh* 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. 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.

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 local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command

rsh otherhost cat remotefile >> localfile

appends the remote file *remotefile* to the localfile *localfile,* while

rsh otherhost cat remotefile ">>" otherremotefile

appends *remotefile* to *otherremotefile.*

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 **rsh** can be omitted.

FILES

/etc/hosts
/usr/hosts/*

SEE ALSO

rlogin(1C)

BUGS

If you are using *csh*(1) and put a *rsh*(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 *rsh* to /dev/null using the **−n** option.

You cannot run an interactive command (like *rogue*(6) or *vi*(1)); use *rlogin*(1C).

Stop signals stop the local *rsh* process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

NAME

rtar, rdd, rmt – remote tape manipulation programs

SYNOPSIS

rtar ...**f**... *host:device* [files] ...

rdd ... [**if**=*host:device*] ... [**of**=*host:device*] ...

rmt [**–f** *host:device*] ...

DESCRIPTION

Rtar, *rdd* and *rmt* are versions of *tar*(1), *dd*(1) and *mt*(1) which work on remote tapes in much the same way as *rdump*(8) and *rrestore*(8). These programs cause another program */etc/rmt* (*rmt*(8)) to be executed on the remote host, which in turn manipulates the specified device according to commands issued by the local program, passing data back and forth as necessary. All keys and options for the remote flavor programs are the same as for the local flavor programs; the only difference is that if a tape device of the form *host:device* is specified, the tape operations are performed on the named device on the named host.

DIAGNOSTICS

Diagnostics are the same as for the local flavor programs, with the exception of a few possible socket errors. Remote errors are reported to the local program.

SEE ALSO

dd(1), mt(1), tar(1), mtio(4), rdump(8), rmt(8), rrestore(8)

BUGS

Rmt is clearly a bad name for 'remote *mt*', since it clashes with */etc/rmt*. It is suggested that one simply install *rmt* as *mt*, since the overhead of using the remote routines is low for this program.

Rtar, *rdd* and *rmt* must all be setuid to root, since they use a privileged socket. The programs know enough to setuid back to the real user id after opening the socket, but there are undoubtedly security problems here.

The program */etc/rmt* has bugs in it. Most of these bugs appear to be handled by *rtar* and friends, but beware.

NAME

ruptime − show host status of local machines

SYNOPSIS

ruptime [−a] [−l] [−t] [−u]

DESCRIPTION

Ruptime gives a status line like *uptime* 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 5 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. The **−l** , **−t** , and **−u** flags specify sorting by load average, uptime, and number of users, respectively.

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 5 minutes then *rwho* assumes the machine is down, and does not report users last known to be logged into that machine.

If a users hasn't typed to the system for a minute or more, then *rwho* reports this idle time. If a user hasn't typed to the system for an hour or more, then the user will be omitted from the output of *rwho* unless the **−a** flag is given.

FILES

/usr/spool/rwho/whod.* information about other machines

SEE ALSO

ruptime(1C), rwhod(8C)

BUGS

This is unwieldy when the number of machines on the local net is large.

NAME

scan – produce a one-line-per-message scan listing

SYNOPSIS

scan [+folder] [msgs] [–ff] [–header] [–help] [–noff] [–noheader]

DESCRIPTION

Scan produces a one-line-per-message listing of the specified messages. Each *scan* line contains the message number (name), the date, the "From" field, the "Subject" field, and, if room allows, some of the body of the message. For example:

```
^ #~^^Date~^^ From~^Subject [Body]
^15+~^^7/ 5~^^Dcrocker~^nned Last week I asked some of
^16 –~^^7/ 5~^^dcrocker~^message id format I recommend
^18~^^7/ 6~^^Obrien~^Re: Exit status from mkdir
^19~^^7/ 7~^^Obrien~^"scan" listing format in MH
```

The '+' on message 15 indicates that it is the current message. The '–' on message 16 indicates that it has been replied to, as indicated by a "Replied:" component produced by an '–annotate' switch to the *repl* command.

If there is sufficient room left on the *scan* line after the subject, the line will be filled with text from the body, preceded by . *Scan* actually reads each of the specified messages and parses them to extract the desired fields. During parsing, appropriate error messages will be produced if there are format errors in any of the messages.

The '–header' switch produces a header line prior to the *scan* listing, and the '–ff' switch will cause a form feed to be output at the end of the *scan* listing.

FILES

^$HOME/mh_profile~^The user profile

PROFILE COMPONENTS

Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder

DEFAULTS

'+folder' defaults to current
'msgs' defaults to all
'–noff'
'–noheader'

CONTEXT

If a folder is given, it will become current. The current message is unaffected.

NAME

screen — repeatedly display output of command on terminal screen

SYNOPSIS

screen [-s <seconds>] [-h] <command> [<arg> ...]

DESCRIPTION

Screen takes its command line (not including any leading flags) and executes it in a subshell. The first screenful of the standard output of the subshell is then displayed on the terminal screen.

The command is repeated after a delay of a few seconds, and the terminal screen is refreshed.

Options are:

-s <seconds>	The delay between the end of one command execution and the start of the next may be given; the default is 15.
-h	The first line of the screen is a header, including the command given, the delay between command executions, and the date and time. If possible, this is in "standout" mode.

AUTHOR

Jeffrey Mogul

DIAGNOSTICS

Few.

BUGS

Many.

NAME

script — make typescript of terminal session

SYNOPSIS

script [**−a**] [file]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript is written to *file*, or appended to *file* if the **−a** option is given. It can be sent to the line printer later with *lpr*. If no file name is given, the typescript is saved in the file *typescript*.

The script ends when the forked shell exits.

This program is useful when using a crt and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply.

BUGS

Script places **everything** in the log file. This is not what the naive user expects.

NAME

sed − stream editor

SYNOPSIS

sed [**−n**] [**−e** script] [**−f** sfile] [file] ...

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **−f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **−e** option and no **−f**'s, the flag **−e** may be omitted. The **−n** option suppresses the default output.

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.

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, '/regular expression/', in the style of *ed*(1) modified thus:

The escape sequence '\n' matches a newline embedded in 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 '!' (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

An argument denoted *text* 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.

An argument denoted *rfile* or *wfile* 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) 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

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 ':' 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 '{') 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 '}' only when the pattern space is selected.

(0) An empty command is ignored.

SEE ALSO

ed(1), grep(1), awk(1), lex(1)

NAME

send – send a message

SYNOPSIS

send [file] [–draft] [–verbose] [–format] [–msgid] [–help] [–noverbose] [–noformat] [–nomsgid]

DESCRIPTION

Send will cause the specified file (default <mh-dir>/draft) to be delivered to each of the addresses in the "To:", "cc:", and "Bcc:" fields of the message. If '–verbose' is specified, *send;* will monitor the delivery of local and net mail. *Send* with no argument will query whether the draft is the intended file, whereas '–draft' will suppress this question. Once the message has been mailed (or queued) successfully, the file will be renamed with a leading comma, which allows it to be retreived until the next draft message is sent. If there are errors in the formatting of the message, *send;* will abort with a (hopefully) helpful error message.

If a "Bcc:" field is encountered, its addresses will be used for delivery, but the "Bcc:" field itself will be deleted from all copies of the outgoing message.

Prior to sending the message, the fields "From: user", and "Date: now" will be prepended to the message. If '–msgid' is specified, then a "Message-Id:" field will also be added to the message. If the message already contains a "From:" field, then a "Sender: user" field will be added instead. (An already existing "Sender:" field will be deleted from the message.)

If the user doesn't specify '–noformat', each of the entries in the "To:" and "cc:" fields will be replaced with "standard" format entries. This standard format is designed to be usable by all of the message handlers on the various systems around the ARPANET.

If an "Fcc: folder" is encountered, the message will be copied to the specified folder in the format in which it will appear to any receivers of the message. That is, it will have the prepended fields and field reformatting.

If a "Distribute-To:" field is encountered, the message is handled as a redistribution message (see *dist* for details), with "Distribution-Date: now" and "Distribution-From: user" added.

FILES

$HOME/mh_profile The user profile

PROFILE COMPONENTS

Path: To determine the user's MH directory

DEFAULTS

'file' defaults to draft
'–noverbose'
'–format'
'–nomsgid'

CONTEXT

Send has no effect on the current message or folder.

NAME

sendbug — mail a system bug report to 4bsd-bugs

SYNOPSIS

sendbug [address]

DESCRIPTION

Bug reports sent to '4bsd-bugs@BERKELEY' are intercepted by a program which expects bug reports to conform to a standard format. *Sendbug* is a shell script to help the user compose and mail bug reports in the correct format. *Sendbug* works by invoking *vi*(1) on a temporary copy of the bug report format outline. The user must fill in the appropriate fields and exit *vi*. *Sendbug* then mails the completed report to '4bsd-bugs@BERKELEY' or the address specified on the command line.

FILES

/usr/ucb/bugformat contains the bug report outline

SEE ALSO

vi(1), sendmail(8)

NAME

sh, for, case, if, while, :, ., break, continue, cd, eval, exec, exit, export, login, read, readonly, set, shift, times, trap, umask, wait – command language

SYNOPSIS

sh [**−ceiknrstuvx**] [arg] ...

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. See **invocation** for the meaning of arguments to the shell.

Commands.

A *simple-command* is a sequence of non blank *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 (see *execve*(2)). 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.

for *name* [**in** *word* ...] **do** *list* **done**
Each time a **for** command is executed *name* is set to the next word in the **for** word list 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**
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.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
The *list* following **if** is executed and if it returns zero the *list* following **then** is executed. Otherwise, the *list* following **elif** is executed and if its value is zero the *list* following **then** is executed. Failing that the **else** *list* is executed.

while *list* [**do** *list*] **done**
A **while** command repeatedly executes the **while** *list* and if its value is zero executes the **do** *list;* otherwise the loop terminates. The value returned by a **while** command is that of the last executed command in the **do** *list.* **until** may be used in place of **while** to negate the loop termination test.

(*list*) Execute *list* in a subshell.

{ *list* } *list* is simply executed.

The following words are only recognized as the first word of a command and when not quoted.

if then else elif fi case in esac for while until do done { }

Command substitution.
The standard output from a command enclosed in a pair of back quotes (``) may be used as part or all of a word; trailing newlines are removed.

Parameter substitution.
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*}
A *parameter* is a sequence of letters, digits or underscores (a *name*), a digit, or any of the characters * @ # ? - $!. 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 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;* the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

$ {*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 *word* is not evaluated unless it is to be used as the substituted string. (So that, for example, echo ${d-'pwd'} will only execute *pwd* if *d* is unset.)

The following *parameters* are automatically set by the shell.

#	The number of positional parameters in decimal.
-	Options supplied to the shell on invocation or by **set**.
?	The value returned by the last executed command in decimal.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following *parameters* are used but not set by the shell.

HOME	The default argument (home directory) for the **cd** command.
PATH	The search path for commands (see **execution**).
MAIL	If this variable is set to the name of a mail file, the shell informs the user of the arrival of mail in the specified file.
PS1	Primary prompt string, by default '$ '.
PS2	Secondary prompt string, by default '> '.
IFS	Internal field separators, normally **space**, **tab**, and **newline**.

Blank interpretation.
After parameter and command substitution, any 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.

File name generation.
Following substitution, 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 /, 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

A character may be *quoted* by preceding it with a \. **\newline** is ignored. All characters enclosed between a pair of quote marks (''), except a single quote, are quoted. Inside double quotes ("") parameter and command substitution occurs and \ quotes the characters \ ' " and $.

"$*" is equivalent to **"$1 $2 ..."** whereas
"$@" is equivalent to **"$1" "$2" ...** .

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 (**$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. 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 (by seeking to the end); otherwise the file is created.

<< *word*
The shell input is read up to a line the same as *word*, or end of file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, **\newline** is ignored, and \ is used to quote the characters \ $ ' and the first character of *word*.

<& *digit*
The standard input is duplicated from file descriptor *digit;* see *dup*(2). Similarly for the standard output using >.

<& − The standard input is closed. Similarly for 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 & 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 is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list; see *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 the user modifies the values of these *parameters* or creates new ones, none of these affects the environment unless the **export** command is used 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.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to *parameters*. Thus these two lines are equivalent

```
TERM=450 cmd args
(export TERM; TERM=450; cmd args)
```

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if the occur after the command name. The following 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. (But 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 shell parameter **$PATH** defines the search path for the directory containing the command. Each alternative directory name is separated by a colon (:). The default path is **:/bin:/usr/bin**. If the command name contains a /, 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 an *a.out* file, it is assumed to be a file containing shell commands. A subshell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Special commands.
The following commands are executed in the shell process and except where specified no input output redirection is permitted for such commands.

: No effect; the command does nothing.

. *file* Read and execute commands from *file* and return. The search path **$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*.

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 non interactive 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 exit from the shell.)

export [*name* ...]
The given names are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of exportable names is printed.

login [*arg* ...]
Equivalent to 'exec login arg ...'.

read *name* ...
One line is read from the standard input; successive words of the input are assigned 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* ...]
The given names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given, a list of all readonly names is printed.

set [**−eknptuvx** [*arg* ...]]

- **−e** If non interactive, exit immediately if a command fails.
- **−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.
- **−** Turn off the **−x** and **−v** options.

These flags can also be used upon invocation of the shell. The current set of flags may be found in **$−**.

Remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, etc. If no arguments are given, the values of all names are printed.

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*] ...
Arg is a command 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. 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 invoked commands. If *n* is 0, the command *arg* is executed on exit from the shell, otherwise upon receipt of signal *n* as numbered in *sigvec*(2). *Trap* with no arguments prints a list of commands associated with each signal number.

umask [*nnn*]
The user file creation mask is set to the octal value *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

wait [*n*]
Wait for the specified process and report its termination status. If *n* is not given, all currently active child processes are waited for. The return code from this command is that of the process waited for.

Invocation.
If the first character of argument zero is −, commands are read from **$HOME/.profile**, if such a file exists. 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, 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 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 *gtty*) then this shell is *interactive*. In this case the terminate signal SIGTERM (see *sigvec*(2)) is ignored (so that 'kill 0' does not kill an interactive shell) and the interrupt signal SIGINT is caught and ignored (so that **wait** is interruptible). In all cases SIGQUIT is ignored by the shell.

The remaining flags and arguments are described under the **set** command.

FILES
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO
csh(1), test(1), execve(2), environ(7)

DIAGNOSTICS
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 then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also **exit**).

BUGS
If << is used to provide standard input to an asynchronous process invoked by &, the shell gets mixed up about naming the input document. A garbage file /tmp/sh* is created, and the shell complains about not being able to find the file by another name.

NAME

shar -- produce shell-script archives

SYNOPSIS

shar files [ ] > archive

DESCRIPTION

Shar is used to collect a number of text files into a single archive, that, when run as a shell script, "extracts" itself to recreate the files.

NAME

show – show (list) messages

SYNOPSIS

show [+folder] [msgs] [–pr] [–nopr] [–draft] [–help] [*l* or *pr* switches]

DESCRIPTION

Show lists each of the specified messages to the standard output (typically, the terminal). The messages are listed exactly as they are, with no reformatting. A program called *l* is invoked to do the listing, and any switches not recognized by *show* are passed along to *l*.

If no "msgs" are specified, the current message is used. If more than one message is specified, *l* will prompt for a <return> prior to listing each message.

l will list each message, a page at a time. When the end of page is reached, *l* will ring the bell and wait for a <RETURN> or <CTRL-D>. If a <return> is entered, *l* will clear the screen before listing the next page, whereas <CTRL-D> will not. The switches to *l* are '–p#' to indicate the page length in lines, and '–w#' to indicate the width of the page in characters.

If the standard output is not a terminal, no queries are made, and each file is listed with a one-line header and two lines of separation.

If '–pr' is specified, then *pr*(l) will be invoked rather than *l*, and the switches (other than '–draft') will be passed along. "Show –draft" will list the file <mh-dir>/draft if it exists.

FILES

$HOME/mh_profile	The user profile
/bin/l	Screen-at-a-time list program
/bin/pr	*pr*(1)

PROFILE COMPONENTS

Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder

DEFAULTS

'+folder' defaults to current
'msgs' defaults to cur
'–nopr'

CONTEXT

If a folder is given, it will become the current message. The last message listed will become the current message.

NAME

size − size of an object file

SYNOPSIS

size [object ...]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in hex and decimal, of each object-file argument. If no file is specified, **a.out** is used.

SEE ALSO

a.out(5)

NAME

size68 — prints sizes of segments in a .b or .68 file

SYNOPSIS

size68 [-hl] file

DESCRIPTION

Size68 prints the sizes of the text, data, and bss segments of a .b or .68 file, in decimal; the total size is also given in both decimal and octal. The -h flag causes the sizes to be given in hexadecimal. The -l flag causes a complete printout in decimal (hex if -h is given) of the values of the 8 header words, namely the magic number, the sizes of the three segments, the size of the symbol table, and the sizes of the relocation commands.

AUTHOR

C.J. Terman

NAME

sleep − suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

Sleep 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

setitimer(2), alarm(3C), sleep(3)

BUGS

Time must be less than 2,147,483,647 seconds.

NAME

soelim − eliminate .so's from nroff input

SYNOPSIS

soelim [file ...]

DESCRIPTION

Soelim reads the specified files or the standard input and performs the textual inclusion implied by the *nroff* directives of the form

.so somefile

when they appear at the beginning of input lines. This is useful since programs such as *tbl* do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of a single minus (−) is taken to be a file name corresponding to the standard input.

Note that inclusion can be suppressed by using ‘’’ instead of ‘.’, i.e.

'so /usr/lib/tmac.s

A sample usage of *soelim* would be

soelim exum?.n | tbl | nroff −ms | col | lpr

SEE ALSO

colcrt(1), more(1)

AUTHOR

William Joy

BUGS

The format of the source commands must involve no strangeness − exactly one blank must precede and no blanks follow the file name.

NAME

sort − sort or merge files

SYNOPSIS

sort [−**mubdfinrt***x*] [+*pos1* [−*pos2*]] ... [−**o** name] [−**T** directory] [name] ...

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name '−' means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

b Ignore leading blanks (spaces and tabs) in field comparisons.

d 'Dictionary' order: only letters, digits and blanks are significant in comparisons.

f Fold upper case letters onto lower case.

i Ignore characters outside the ASCII range 040-0176 in nonnumeric comparisons.

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. Option **n** implies option **b**.

r Reverse the sense of comparisons.

t*x* 'Tab character' separating fields is *x*.

The notation +*pos1* −*pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m.n*, optionally followed by one or more of the flags **bdfinr**, where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first nonblank in the field; **b** is attached independently to *pos2*. A missing *.n* means .0; a missing −*pos2* means the end of the line. Under the −**t***x* option, fields are strings separated by *x*; otherwise fields are nonempty nonblank strings separated by blanks.

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.

These option arguments are also understood:

c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

m Merge only, the input files are already sorted.

o The next argument 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.

T The next argument is the name of a directory in which temporary files should be made.

u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

EXAMPLES

Print in alphabetical order all the unique spellings in a list of words. Capitalized words differ from uncapitalized.

```
sort −u +0f +0 list
```

Print the password file (*passwd*(5)) sorted by user id number (the 3rd colon-separated field).

```
sort -t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month day) entries. The options **-um** with just one input file make the choice of a unique representative from a set of equal lines predictable.

```
sort -um +0 -1 dates
```

FILES

/usr/tmp/stm*, /tmp/* first and second tries for temporary files

SEE ALSO

uniq(1), comm(1), rev(1), join(1)

DIAGNOSTICS

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option **-c**.

BUGS

Very long lines are silently truncated.

NAME

sortbib − sort bibliographic database

SYNOPSIS

sortbib [−sKEYS] database ...

DESCRIPTION

Sortbib sorts files of records containing *refer* key-letters by user-specified keys. Records may be separated by blank lines, or by .[and .] delimiters, but the two styles may not be mixed together. This program reads through each *database* and pulls out key fields, which are sorted separately. The sorted key fields contain the file pointer, byte offset, and length of corresponding records. These records are delivered using disk seeks and reads, so *sortbib* may not be used in a pipeline to read standard input.

By default, *sortbib* alphabetizes by the first %A and the %D fields, which contain the senior author and date. The −s option is used to specify new *KEYS*. For instance, −sATD will sort by author, title, and date, while −sA+D will sort by all authors, and date. Sort keys past the fourth are not meaningful. No more than 16 databases may be sorted together at one time. Records longer than 4096 characters will be truncated.

Sortbib sorts on the last word on the %A line, which is assumed to be the author's last name. A word in the final position, such as "jr." or "ed.", will be ignored if the name beforehand ends with a comma. Authors with two-word last names or unusual constructions can be sorted correctly by using the *nroff* convention "\0" in place of a blank. A %Q field is considered to be the same as %A, except sorting begins with the first, not the last, word. *Sortbib* sorts on the last word of the %D line, usually the year. It also ignores leading articles (like "A" or "The") when sorting by titles in the %T or %J fields; it will ignore articles of any modern European language. If a sort-significant field is absent from a record, *sortbib* places that record before other records containing that field.

SEE ALSO

refer(1), addbib(1), roffbib(1), indxbib(1), lookbib(1)

AUTHORS

Greg Shenaut, Bill Tuthill

BUGS

Records with missing author fields should probably be sorted by title.

NAME

spell, spellin, spellout − find spelling errors

SYNOPSIS

spell [**−v**] [**−b**] [**−x**] [**−d** hlist] [**−s** hstop] [**−h** spellhist] [file] ...

spellin [list]

spellout [**−d**] list

DESCRIPTION

Spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes 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, tbl* and *eqn*(1) constructions.

Under the **−v** option, all words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

Under the **−b** option, British spelling is checked. Besides preferring *centre, colour, speciality, travelled,* etc., this option insists upon *-ise* in words like *standardise,* Fowler and the OED to the contrary notwithstanding.

Under the **−x** option, every plausible stem is printed with '**=**' for each word.

The spelling list is based on many sources. While it is more haphazard than an ordinary dictionary, it is also more effective with proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

The auxiliary files used for the spelling list, stop list, and history file may be specified by arguments following the **−d**, **−s**, and **−h** options. The default files are indicated below. Copies of all output may be accumulated in the history file. The stop list filters out misspellings (e.g. thier=thy−y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by *spell.* Both expect a set of words, one per line, from the standard input. *Spellin* combines the words from the standard input and the preexisting *list* file and places a new list on the standard output. If no *list* file is specified, the new list is created from scratch. *Spellout* looks up each word from the standard input and prints on the standard output those that are missing from (or present on, with option **−d**) the hashed *list* file. For example, to verify that *hookey* is not on the default spelling list, add it to your own private list, and then use it with *spell*,

```
echo hookey | spellout /usr/dict/hlista
echo hookey | spellin /usr/dict/hlista > myhlist
spell −d myhlist huckfinn
```

FILES

/usr/dict/hlist[ab]	hashed spelling lists, American & British, default for **−d**
/usr/dict/hstop	hashed stop list, default for **−s**
/dev/null	history file, default for **−h**
/tmp/spell.$$*	temporary files
/usr/lib/spell	

SEE ALSO

deroff(1), sort(1), tee(1), sed(1)

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions.
British spelling was done by an American.

NAME

spline — interpolate smooth curve

SYNOPSIS

spline [option] ...

DESCRIPTION

Spline takes pairs of numbers from the standard input as abcissas 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 (R. W. Hamming, *Numerical Methods for Scientists and Engineers,* 2nd ed., 349ff) 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. By 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 abcissas start at lower limit (default 0).

SEE ALSO

graph(1G), plot(1G)

DIAGNOSTICS

When data is not strictly monotone in x, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1000 input points is enforced silently.

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), as many as necessary, onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically. 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.

NAME

strings − find the printable strings in a object, or other binary, file

SYNOPSIS

strings [−] [−**o**] [−*number*] file ...

DESCRIPTION

Strings looks for ascii strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null. Unless the − flag is given, *strings* only looks in the initialized data space of object files. If the −**o** flag is given, then each string is preceded by its offset in the file (in octal). If the −*number* flag is given then number is used as the minimum string length rather than 4.

Strings is useful for identifying random object files and many other things.

SEE ALSO

od(1)

BUGS

The algorithm for identifying strings is extremely primitive

NAME

strip − remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space after a program has been debugged.

The effect of *strip* is the same as use of the **−s** option of *ld*.

FILES

/tmp/stm? temporary file

SEE ALSO

ld(1)

NAME

struct − structure Fortran programs

SYNOPSIS

struct [option] ... file

DESCRIPTION

Struct translates the Fortran program specified by *file* (standard input default) into a Ratfor program. Wherever possible, Ratfor control constructs replace the original Fortran. Statement numbers appear only where still necessary. Cosmetic changes are made, including changing Hollerith strings into quoted strings and relational operators into symbols (.e.g. ".GT." into ">"). The output is appropriately indented.

The following options may occur in any order.

−s Input is accepted in standard format, i.e. comments are specified by a c, C, or * in column 1, and continuation lines are specified by a nonzero, nonblank character in column 6. Normally input is in the form accepted by *f77*(1)

−i Do not turn computed goto statements into switches. (Ratfor does not turn switches back into computed goto statements.)

−a Turn sequences of else ifs into a non-Ratfor switch of the form

```
switch
        {   case pred1: code
            case pred2: code
            case pred3: code
            default: code
        }
```

The case predicates are tested in order; the code appropriate to only one case is executed. This generalized form of switch statement does not occur in Ratfor.

−b Generate goto's instead of multilevel break statements.

−n Generate goto's instead of multilevel next statements.

−t*n* Make the nonzero integer *n* the lowest valued label in the output program (default 10).

−c*n* Increment successive labels in the output program by the nonzero integer *n* (default 1).

−e*n* If *n* is 0 (default), place code within a loop only if it can lead to an iteration of the loop. If *n* is nonzero, admit a small code segments to a loop if otherwise the loop would have exits to several places including the segment, and the segment can be reached only from the loop. 'Small' is close to, but not equal to, the number of statements in the code segment. Values of n under 10 are suggested.

FILES

/tmp/struct*
/usr/lib/struct/*

SEE ALSO

f77(1)

BUGS

Struct knows Fortran 66 syntax, but not full Fortran 77.
If an input Fortran program contains identifiers which are reserved words in Ratfor, the structured version of the program will not be a valid Ratfor program.
The labels generated cannot go above 32767.
If you get a goto without a target, try **−e** .

NAME

stty − set terminal options

SYNOPSIS

stty [option ...]

DESCRIPTION

Stty sets certain I/O options on the current output terminal, placing its output on the diagnostic output. With no argument, it reports the speed of the terminal and the settings of the options which are different from their defaults. With the argument "all", all normally used option settings are reported. With the argument "everything", everything *stty* knows about is printed. The option strings are selected from the following set:

even allow even parity input
−even disallow even parity input
odd allow odd parity input
−odd disallow odd parity input
raw raw mode input (**no** input processing (erase, kill, interrupt, ...); parity bit passed back)
−raw negate raw mode
cooked same as '−raw'
cbreak make each character available to *read*(2) as received; no erase and kill processing, but all other processing (interrupt, suspend, ...) is performed
−cbreak make characters available to *read* only when newline is received
−nl allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl accept only new-line to end lines
echo echo back every character typed
−echo do not echo characters
lcase map upper case to lower case
−lcase do not map case
tandem enable flow control, so that the system sends out the stop character when its internal queue is in danger of overflowing on input, and sends the start character when it is ready to accept further input
−tandem disable flow control
−tabs replace tabs by spaces when printing
tabs preserve tabs
ek set erase and kill characters to # and @

For the following commands which take a character argument *c*, you may also specify *c* as the "u" or "undef", to set the value to be undefined. A value of "^x", a 2 character sequence, is also interpreted as a control character, with "^?" representing delete.

erase *c* set erase character to *c* (default '#', but often reset to ^H.)
kill *c* set kill character to *c* (default '@', but often reset to ^U.)
intr *c* set interrupt character to *c* (default DEL or ^? (delete), but often reset to ^C.)
quit *c* set quit character to *c* (default control \.)
start *c* set start character to *c* (default control Q.)
stop *c* set stop character to *c* (default control S.)
eof *c* set end of file character to *c* (default control D.)
brk *c* set break character to *c* (default undefined.) This character is an extra wakeup causing character.
cr0 cr1 cr2 cr3
select style of delay for carriage return (see *ioctl*(2))
nl0 nl1 nl2 nl3
select style of delay for linefeed
tab0 tab1 tab2 tab3

select style of delay for tab
ff0 ff1 select style of delay for form feed
bs0 bs1 select style of delay for backspace

tty33 set all modes suitable for the Teletype Corporation Model 33 terminal.
tty37 set all modes suitable for the Teletype Corporation Model 37 terminal.
vt05 set all modes suitable for Digital Equipment Corp. VT05 terminal
dec set all modes suitable for Digital Equipment Corp. operating systems users; (erase, kill, and interrupt characters to ^?, ^U, and ^C, decctlq and "newcrt".)

tn300 set all modes suitable for a General Electric TermiNet 300
ti700 set all modes suitable for Texas Instruments 700 series terminal
tek set all modes suitable for Tektronix 4014 terminal
0 hang up phone line immediately
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb
Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).

A teletype driver which supports the job control processing of *csh*(1) and more functionality than the basic driver is fully described in *tty*(4). The following options apply only to it.

new Use new driver (switching flushes typeahead).
crt Set options for a CRT (crtbs, ctlecho and, if >= 1200 baud, crterase and crtkill.)
crtbs Echo backspaces on erase characters.
prterase For printing terminal echo erased characters backwards within "\" and "/".
crterase Wipe out erased characters with "backspace-space-backspace."
−crterase Leave erased characters visible; just backspace.
crtkill Wipe out input on like kill ala **crterase.**
−crtkill Just echo line kill character and a newline on line kill.
ctlecho Echo control characters as "^*x*" (and delete as "^?".) Print two backspaces following the EOT character (control D).
−ctlecho Control characters echo as themselves; in cooked mode EOT (control-D) is not echoed.

decctlq After output is suspended (normally by ^S), only a start character (normally ^Q) will restart it. This is compatible with DEC's vendor supplied systems.

−decctlq After output is suspended, any character typed will restart it; the start character will restart output without providing any input. (This is the default.)
tostop Background jobs stop if they attempt terminal output.
−tostop Output from background jobs to the terminal is allowed.
tilde Convert "~" to "`" on output (for Hazeltine terminals).
−tilde Leave poor "~" alone.
flusho Output is being discarded usually because user hit control O (internal state bit).
−flusho Output is not being discarded.
pendin Input is pending after a switch from cbreak to cooked and will be re-input when a read becomes pending or more input arrives (internal state bit).
−pendin Input is not pending.
intrup Send a signal (SIGTINT) to the terminal control process group whenever an input record (line in cooked mode, character in cbreak or raw mode) is available for reading.
−intrup Don't send input available interrupts.
mdmbuf Start/stop output on carrier transitions (not implemented).
−mdmbuf
Return error if write attempted after carrier drops.
litout Send output characters without any processing.

−litout Do normal output processing, inserting delays, etc.
nohang Don't send hangup signal if carrier drops.
−nohang Send hangup signal to control process group when carrier drops.
etxack Diablo style etx/ack handshaking (not implemented).

The following special characters are applicable only to the new teletype driver and are not normally changed.

susp *c* set suspend process character to *c* (default control Z).
dsusp *c* set delayed suspend process character to *c* (default control Y).
rprnt *c* set reprint line character to *c* (default control R).
flush *c* set flush output character to *c* (default control O).
werase *c* set word erase character to *c* (default control W).
lnext *c* set literal next character to *c* (default control V).

SEE ALSO

ioctl(2), tabs(1), tset(1), tty(4)

NAME

style − analyze surface characteristics of a document

SYNOPSIS

style [**−ml**] [**−mm**] [**−a**] [**−e**] [**−l** num] [**−r** num] [**−p**] [**−P**] file ...

DESCRIPTION

Style analyzes the surface characteristics of the writing style of a document. It reports on readability, sentence length and structure, word length and usage, verb type, and sentence openers. Because *style* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package **−ms** may be overridden with the flag **−mm**. The flag **−ml**, which causes **deroff** to skip lists, should be used if the document contains many lists of non-sentences. The other options are used to locate sentences with certain characteristics.

−a print all sentences with their length and readability index.

−e print all sentences that begin with an expletive.

−p print all sentences that contain a passive verb.

−l*num* print all sentences longer than *num*.

−r*num* print all sentences whose readability index is greater than *num*.

−P print parts of speech of the words in the document.

SEE ALSO

deroff(1), diction(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks.

NAME

su – substitute user id temporarily

SYNOPSIS

su [userid]

DESCRIPTION

Su demands the password of the specified *userid,* and if it is given, changes to that *userid* and invokes the Shell *sh*(1) without changing the current directory. The user environment is unchanged except for HOME and SHELL, which are taken from the password file for the user being substituted (see *environ*(7)). The new user ID stays in force until the Shell exits.

If no *userid* is specified, 'root' is assumed. To remind the super-user of his responsibilities, the Shell substitutes '#' for its usual prompt.

SEE ALSO

sh(1)

BUGS

Local administrative rules cause restrictions to be placed on who can *su* to 'root', even with the root password. These rules vary from site to site.

NAME

sum – sum and count blocks in a file

SYNOPSIS

sum 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.

SEE ALSO

wc(1)

DIAGNOSTICS

'Read error' is indistinguishable from end of file on most devices; check the block count.

NAME

symlchk – check for bad symbolic links

SYNOPSIS

symlchk [–p] files [. . .]

DESCRIPTION

Symbolic links, unlike hard links, can point to files that do not exist. The *symlchk* program takes a list of files (possibly directories) and recursively descends through them, checking for symbolic links that cannot be followed.

Normally, bad links are printed. If the –p flag is given, all symbolic links are printed.

DIAGNOSTICS

Exit status is < 0 if the program fails, 0 if no bad links are found, otherwise the number of bad links.

AUTHOR

Jeffrey Mogul

SEE ALSO

ls(1), ln(1), symlink(2), readlink(2), stat(2)

BUGS

Even good links that point to directories are not followed in the recursive descent. This may not be a bug, since it would complicate the task of checking the entire file system, for example.

A symbolic link may appear bad because the user is denied access, even though the link is actually good. Running this program as root will avoid this.

NAME

symorder — rearrange name list

SYNOPSIS

symorder orderlist symbolfile

DESCRIPTION

Orderlist is a file containing symbols to be found in symbolfile, 1 symbol per line.

Symbolfile is updated in place to put the requested symbols first in the symbol table, in the order specified. This is done by swapping the old symbols in the required spots with the new ones. If all of the order symbols are not found, an error is generated.

This program was specifically designed to cut down on the overhead of getting symbols from /vmunix.

SEE ALSO

nlist(3)

NAME

sysline – display system status on status line of a terminal

SYNOPSIS

sysline [–bcdehDilmpqrsj] [+N]

DESCRIPTION

sysline runs in the background and periodically displays system status information on the status line of the terminal. Not all terminals contain a status line. Those that do include the h19, concept 108, Ann Arbor Ambassador, vt100, Televideo 925/950 and Freedom 100. If no flags are given, *sysline* displays the time of day, the current load average, the change in load average in the last 5 minutes, the number of users (followed by a 'u'), the number of runnable process (followed by a 'r')[VAX only], the number of suspended processes (followed by a 's')[VAX only], and the users who have logged on and off since the last status report. Finally, if new mail has arrived, a summary of it is printed. If there is unread mail in your mailbox, an asterisk will appear after the display of the number of users. The display is normally in reverse video (if your terminal supports this in the status line) and is right justified to reduce distraction. Every fifth display is done in normal video to give the screen a chance to rest.

If you have a file named .who in your home directory, then the contents of that file is printed first. One common use of this feature is to alias chdir, pushd, and popd to place the current directory stack in ~/.who after it changes the new directory.

The following flags may be given on the command line.

–b Beep once every half hour and twice every hour, just like those obnoxious watches you keep hearing.

–c Clear the status line for 5 seconds before each redisplay.

–d Debug mode -- print status line data in human readable format

–e Print out only the information. Do not print out the control commands necessary to put the information on the bottom line. This option is useful for putting the output of *sysline* onto the mode line of an emacs window.

–D Print out the current day/date before the time.

–h Print out the host machine's name after the time [VAX only].

–l Don't print the names of people who log in and out.

–m Don't check for mail.

–p Don't report the number of process which are runnable and suspended.

–r Don't display in reverse video.

+N Update the status line every N seconds. The default is 60 seconds.

–q Don't print out diagnostic messages if something goes wrong when starting up.

–i Print out the process id of the *sysline* process onto standard output upon startup. With this information you can send the alarm signal to the *sysline* process to cause it to update immediately. *sysline* writes to the standard error, so you can redirect the standard output into a file to catch the process id.

–s Print "short" form of line by left-justifying *iff* escapes are not allowed in the status line. Some terminals (the Televideos and Freedom 100 for example) do not allow cursor movement (or other "intelligent" operations) in the status line. For these terminals, *sysline* normally uses blanks to cause right-justification. This flag will disable the adding of the blanks.

–j Force the sysline output to be left justified even on terminals capable of cursor

movement on the status line.

If you have a file .syslinelock in your home directory, then *sysline* will not update its statistics and write on your screen, it will just go to sleep for a minute. This is useful if you want to momentarily disable *sysline*. Note that it may take a few seconds from the time the lock file is created until you are guaranteed that *sysline* will not write on the screen.

FILES

/etc/utmp	names of people who are logged in
/dev/kmem	contains process table [VAX only]
${HOME}/.who	information to print on bottom line
${HOME}/.syslinelock	when it exists, sysline will not print

AUTHORS

John Foderaro
Tom Ferrin converted it to use termcap.
Mark Horton added terminfo capability.

BUGS

If you interrupt the display then you may find your cursor missing or stuck on the status line. The best thing to do is reset the terminal.
If there is too much for one line, the excess is thrown away.

NAME

tabs − set terminal tabs

SYNOPSIS

tabs [**−n**] [terminal]

DESCRIPTION

Tabs sets the tabs on a variety of terminals. Various terminal names given in *term*(7) are recognized; the default is, however, suitable for most 300 baud terminals. If the **−n** flag is present then the left margin is not indented as is normal.

SEE ALSO

stty(1), term(7)

BUGS

It's much better to use *tset*(1).

NAME

tail − deliver the last part of a file

SYNOPSIS

tail [±number[**lbc**][**fr**]] [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. *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.

Specifying **r** causes tail to print lines from the end of the file in reverse order. The default for **r** is to print the entire file this way. Specifying **f** causes *tail* to not quit at end of file, but rather wait and try to read repeatedly in hopes that the file will grow.

SEE ALSO

dd(1)

BUGS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length.

Various kinds of anomalous behavior may happen with character special files.

NAME

talk — talk to another user

SYNOPSIS

talk person [ttyname]

DESCRIPTION

Talk is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on you own machine, then *person* is just the person's login name. If you wish to talk to a user on another host, then *person* is of the form :

host!user or
host.user or
host:user or
user@host

though *host@user* is perhaps preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, it sends the message

Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine

to the user you wish to talk to. At this point, the recipient of the message should reply by typing

talk your_name@your_machine

It doesn't matter from which machine the recipient replies, as long as his login-name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing control L will cause the screen to be reprinted, while your erase, kill, and word kill characters will work in talk as normal. To exit, just type your interrupt character; *talk* then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the *mesg* command. At the outset talking is allowed. Certain commands, in particular *nroff* and *pr*(1) disallow messages in order to prevent messy output.

FILES

/etc/hosts to find the recipient's machine
/etc/utmp to find the recipient's tty

SEE ALSO

mesg(1), who(1), mail(1), write(1)

NAME

tangle, weave – convert web file into pascal file, tex file

SYNOPSIS

tangle webfile[.web] [changefile[.ch]]
weave [**–x**] webfile[.web] [changefile[.ch]]

DESCRIPTION

The *tangle* program converts a WEB source document into a Pascal program that may be compiled in the usual way with the on-line Pascal compiler (e.g., *pc*(1)). The output file is all in lower case and packed into lines of 72 characters or less, with the only concession to readability being the terminatation of lines at semicolons when this can be done conveniently.

WEB is a new language that Don Knuth has created, primarily for his own use in rewriting TeX. The WEB language allows one to prepare a single document that contains all the information that is needed both to produce a compilable Pascal program and to produce a well-formatted document describing the program in as much detail as the writer may desire. The user of WEB must be familiar with both TeX and Pascal.

WEB provides a relatively simple, although adequate, macro facility that permits a Pascal program to be written in small easily-understood modules. The *tangle* program assembles these modules into a usable Pascal program. The command line should have either one or two names on it. The first is taken as the WEB file (and .web is added if there is no extension). If there is another name, it is a change file (and .ch is added if there is no extension). The change file overides parts of the WEB file, as described in the documentation.

The output files are a Pascal file and a string pool file, whose names are formed by adding .p and .pool respectively to the root of the WEB file name.

The *weave* program is used to create a TeX file for viewing the WEB program. It takes appropriate care of typographic details like page layout and the use of indentation, italics, boldface, etc.,and it supplies extensive cross-index information that it gathers automatically. The command line arguments are the same as for *tangle* except for the options. The –x option says to omit the index, module name list, and table of contents pages. (A CONTENTS.tex file will still be written, however, unless some default webhdr macros are redefined.)

The output TeX file name is formed by using .tex as the extension of the WEB file name.

There are several macros in webhdr.tex that probably should be redefined by the user at the beginning of the WEB file. It is a good idea to set \title to the name of the program. And, to cause output of only changed modules, one can say \let\maybe=\iffalse.

WEB and the November 1981 versions of *tangle and weave* are described in a hard-copy document available from Phyllis Winkler in MJH 326. This document was, of course, written in WEB and it speaks well for the new language. Get a copy if you think that you might like to use WEB.

FILES

/usr/stanford/lib/tex82/macros/webhdr.tex TeX macros used by weave output.

SEE ALSO

tex(1)
pc(1)
pxp(1) (for formatting tangle output when debugging)

BUGS

There probably should be some way to put the output on a different directory from the WEB file.

AUTHORS

WEB was designed by Donald E. Knuth, who implemented it using itself. Installed at Stanford by Howard Trickey, November 30, 1982.

NAME

tar — tape archiver

SYNOPSIS

tar [key] [filename filename ...] [-]

DESCRIPTION

Tar saves multiple files into a single archive file, or else unpacks a single archive file into a set of destination files. It was written for use in copying disk files to tape, and restoring them to disk from tape (hence its name), but in fact it works just fine with the archive a disk file or even a pipe.

The behavior of *Tar* is determined by the *key* argument. The *key* is a string of characters containing a function letter (telling *tar* whether to read or write the archive file), and possibly one or more function modifiers that tell it such things as whether or not it is supposed to print a list of files that it is processing, and whether it is supposed to honor the file owner and protection information when files are being restored from an archive. The remaining arguments after the *key* are file or directory names specifying the files to dump or restore. If the key contains the letter "f", then the first argument after the key is the name of a file to use as the archive.

In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

When tar is dumping files, a filename argument of "−" (a single dash) causes *tar* to read and process a list of filenames from its standard input, then resume taking filenames from the command line after the end of standard input is reached. Thus

```
tar c Apples fr* cherries
```

means the same thing as

```
ls -d fr* | tar c Apples - cherries
```

The function portion of the key is one of the following letters:

r Create an archive. If the archive is a magnetic tape, then skip to the end of the existing files on the tape before writing. All of the files named in the arguments to *tar* will be combined into a single archive file, and that archive will be appended to the end of the tape. When writing archives to media other than tape, you should use the "c" function and not the "r" function.

u Update an archive. Each of the files named in the arguments to *tar* will be compared to the existing contents of the archive. If the named file is not already in the archive, or if it has been updated since it was last placed into the archive, then it will be appended to the end of the archive.

c Create an archive. If the archive is a magnetic tape, it will be rewound and overwritten. If the archive is a disk file, it will be overwritten. All of the files named in the arguments to *tar* will be combined into a single archive file. To append to the end of an existing magnetic tape archive, instead of overwriting it, you should use the "r" option and not the "c" option.

x Extract. Read an archive. Every file that is both stored in the archive and whose name is given as an argument to *tar* will be read from the archive and created on disk. If no file argument is given, the entire content of the tape is extracted.

The name that extracted files are given on disk is always relative to the current working directory, unless the files in the archive were saved with absolute names. Thus, if an archive was written with the command

```
cd /usr
tar c .
```

then it can be restored as /tmp/usr with the command

```
cd /tmp/usr
```

tar x .

because the "." is a relative filename. However, if the archive was written with the command

tar c /usr

then the command

cd /tmp/usr
tar x .

Would do nothing, because the name of the file in the archive is not "." but "/usr". To restore it you would need to type

tar x /usr

or

tar x

which would overwrite the existing /usr; probably with drastic consequences. Use the "a" option (below) to help read tapes that were written with absolute pathnames.

The numeric owner and group information, modification time, and mode bits are restored (if possible). If multiple entries specifying the same file are in the archive, the last one overwrites all earlier.

t Make a table of contents of the archive. If no file argument is given, all of the names in the archive are listed. If file arguments are given, then only names in the archive whose pathname begins with one of the file arguments will be listed. It is a good idea to look at an archive with "tar t" before restoring from it, just to make sure that the file names in the archive are not absolute.

o When writing an archive, tar normally includes information specifying owner and modes of directories in the archive. Former versions of tar, when encountering this information will give error message of the form

"<name>/: cannot create".

The "o" option will suppress the directory information.

p The "p" option says to restore files to their original modes, ignoring the present *umask*(2). Setuid and sticky information will also be restored to the super-user.

The following characters may be used in addition to those listed above.

0, ..., 9 This modifier selects an alternate drive on which the tape is mounted. The default is drive 0 at 1600 bpi, which is normally /dev/rmt8.

v Normally *tar* does its work silently. The **v** (verbose) option make *tar* type the name of each file it treats preceded by the function letter. With the **t** function, the verbose option gives more information about the tape entries than just their names.

w *Tar* prints the action to be taken followed by file name, then wait for user confirmation. If a word beginning with 'y' is given, the action is done. Any other input means don't do it.

f *Tar* uses the next argument as the name of the archive instead of /dev/rmt?. For example, the command

tar cvf /tmp/demo.tar abc def

will create an archive named /tmp/demo.tar, whose contents are the files "abc" and "def" in the current working directory. The "v" will cause it to print the name of each file as it is being copied. If the name of the file is '−', tar writes to standard output or reads from standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a filter chain. *Tar* can thus be used to move hierarchies with the command

cd fromdir; tar cf - . | (cd todir; tar xf -)

b *Tar* uses the next argument as the blocking factor for tape records. The default is 20 (the maximum). This option should only be used with raw magnetic tape archives (See **f** above). The block size is determined automatically when reading tapes (key letters 'x' and 't').

l tells *tar* to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.

m tells *tar* not to restore the modification times. The modification time will be the time of extraction.

h Force *tar* to follow symbolic links as if they were normal files or directories. Normally, *tar* does not follow symbolic links.

B Forces input and output blocking to 20 blocks per record. This option was added so that *tar* can work across a communications channel where the blocking may not be maintained.

F A kludgey option that tells *tar* to ignore certain files when creating archives. As best as we can determine by reading the code, the files ignored are: directories named "SCCS", files named "core" or "errs" or whose names are less than 3 characters long. If the flag is given more than once (double kludge?) then additionally files whose names end in ".o", and files named "a.out", are also ignored.

i Normally, if *tar* sees a bad checksum (perhaps from a tape error) it will give up. This flag tells *tar* to keep reading the archive in the hope that the error is temporary.

a Tells *tar* to drop a leading "/", if present, of the file names it reads from an archive. This is handy if you get a tape with absolute filenames (however, see the description below of the **−C** flag; it is antisocial to write tapes with absolute filenames.)

If a file name is preceded by **−C**, then *tar* will perform a *chdir*(2) to that file name. This allows multiple directories not related by a close common parent to be archived using short relative path names. For example, to archive files from /usr/include and from /etc, one might use

```
tar c -C /usr include -C / etc
```

Previous restrictions dealing with *tar*'s inability to properly handle blocked archives have been lifted.

FILES

/dev/rmt?
/tmp/tar*

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

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 current limit on file name length is 100 characters.
There is no way to selectively follow symbolic links.

NAME

tbl − format tables for nroff or troff

SYNOPSIS

tbl [files] ...

DESCRIPTION

Tbl is a preprocessor for formatting tables for *nroff* or *troff*(1). The input files are copied to the standard output, except for lines between and are reformatted. Details are given in the *tbl*(1) reference manual.

EXAMPLE

As an example, letting \t represent a tab (which should be typed as a genuine tab) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

Household Population		
Town	Households	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Branchburg	1644	3.49
Bridgewater	7897	3.81
Far Hills	240	3.19

If no arguments are given, *tbl* reads the standard input, so it may be used as a filter. When *tbl* is used with *eqn* or *neqn* the *tbl* command should be first, to minimize the volume of data passed through pipes.

SEE ALSO

troff(1), eqn(1)

M. E. Lesk, *TBL*.

NAME

tc − photoypesetter simulator

SYNOPSIS

tc [−t] [−sN] [−pL] [file]

DESCRIPTION

Tc interprets its input (standard input default) as device codes for a Graphic Systems phototypesetter (cat). The standard output of *tc* is intended for a Tektronix 4015 (a 4014 terminal with ASCII and APL character sets). The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, using overstruck combinations where necessary. Typical usage:

troff −t file | tc

At the end of each page *tc* waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command **e** will suppress the screen erase before the next page; **s**N will cause the next N pages to be skipped; and **!**line will send line to the shell.

The command line options are:

−t Don't wait between pages; for directing output into a file.

−sN Skip the first N pages.

−pL Set page length to L. L may include the scale factors **p** (points), **i** (inches), **c** (centimeters), and **P** (picas); default is picas.

'*−l w*' Multiply the default aspect ratio, 1.5, of a displayed page by *l*/*w*.

SEE ALSO

troff(1), plot(1G)

BUGS

Font distinctions are lost.
tc's character set is limited to ASCII in just one size.
The aspect ratio option is unbelievable.

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*. Option **−i** ignores interrupts; option **−a** causes the output to be appended to the *files* rather than overwriting them.

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 input mode. In this mode, text typed is sent to the remote host. To issue *telnet* commands when in input mode, precede them with the *telnet* "escape character" (initially "^["). When in command mode, the normal terminal editing conventions are available.

The following commands are available. Only enough of each command to uniquely identify it need be typed.

open *host* [*port*]
Open a connection to the named host. If the 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*(5)) or an Internet address specified in the "dot notation".

close Close a TELNET session and return to command mode.

quit Close any open TELNET session and exit *telnet*.

z Suspend *telnet*. This command only works when the user is using the *csh*(1).

escape [*escape-char*]
Set the *telnet* "escape character". Control characters may be specified as "^" followed by a single letter; e.g. "control-X" is "^X".

status Show the current status of *telnet*. This includes the peer one is connected to, as well as the state of debugging.

options
Toggle viewing of TELNET options processing. When options viewing is enabled, all TELNET option negotiations will be displayed. Options sent by *telnet* are displayed as "SENT", while options received from the TELNET server are displayed as "RCVD".

crmod Toggle carriage return mode. When this mode is enabled any carriage return characters received from the remote host will be mapped into a carriage return and a line feed. This mode does not affect those characters typed by the user, only those received. This mode is not very useful, but is required for some hosts that like to ask the user to do local echoing.

? [*command*]
Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information available about the command only.

BUGS

This implementation is very simple because *rlogin*(1C) is the standard mechanism used to communicate locally with hosts.

NAME

test − condition command

SYNOPSIS

test expr

DESCRIPTION

test evaluates the expression *expr*, and if its value is true then returns zero exit status; otherwise, a non zero exit status is returned. *test* returns a non zero exit if there are no arguments.

The following primitives are used to construct *expr*.

−r file true if the file exists and is readable.

−w file true if the file exists and is writable.

−f file true if the file exists and is not a directory.

−d file true if the file exists exists and is a directory.

−s file true if the 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 nonzero.

s1 **=** s2 true if the strings *s1* and *s2* are equal.

s1 **!=** s2 true if the strings *s1* and *s2* are not equal.

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**, or **−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

(expr **)**
parentheses for grouping.

−a has higher precedence than **−o**. Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the Shell and must be escaped.

SEE ALSO

sh(1), find(1)

NAME

tex, latex, initex, virtex -- text formatting and typesetting

SYNOPSIS

tex [first line]

latex [first line]

initex [first line]

virtex [first line]

DESCRIPTION

TeX formats interspersed text and commands and outputs a typesetter independent file (called *DVI* which is short for *DeVice Independent*). The TeX language is described in the publication *The TeXbook* by Donald E. Knuth. Old manuals *(TeX and METAFONT: New Directions in Typesetting)* should be put away where they will never be seen, since there are literally hundereds of differences from the old TeX78 system that it describes.

Any arguments given on the command line to the TeX programs are passed to them as the first input line. As described in the TeXbook, that line should begin with a file name or a \controlsequence. The normal usage is to say "**tex** *paper*" to start processing *paper.tex*. The name "paper" will be the "jobname", and is used in forming output file names. Directory paths are stripped from "paper" before forming the jobname, so all output will be in the current directory. If TeX doesn't get a file name in the first line, the jobname is "texput". The default '.tex' extension can be overridden by specifying an extension explicitly.

If there is no paper.tex in the current directory, TeX will look through a search path of directories to try to find it. The standard library on the default search path has the basic format package, plain.tex, described in the TeXbook, as well as several others. Note that it is hardly ever necessary to \input plain, since the *tex* program has preloaded it. This means that all of the control sequences discussed in the TeXbook are known to TeX. Another TeX processor in common use is *latex*. It is like tex in all ways except that it has the format package *latex.tex* preloaded instead of plain.tex. This is a Scribe-inspired document preparation system being developed by Leslie Lamport. See the manual page *latex*(1) for more information.

The output DVI file is written on *name.dvi* where *name* is the jobname. A log of error messages goes into *name.log*.

DVI files should be printed on the Dover with *dvip* (1). Note that there have been incompatible changes in the DVI format between TeX78 and TeX, so programs used to print TeX78 output will not work for TeX.

A note about fonts: the TeXbook says that Plain TeX uses the basic fonts cmr10, cmbx10, etc. In fact it uses amr10, ambx10, etc., and whenever the TeXbook implies that you should use a font name beginning with cm, you should substitute one beginning with am. (Cm stands for "Computer Modern", which hasn't been designed yet. Am stands for "Almost Computer Modern", the interim font family. A further source of confusion is that there actually are some cm fonts installed, but these are for use by the obsolete TeX78 and are slightly incompatible with TeX.)

There are some environment variables that can be used to set up directory paths to search when TeX opens a file for input. For example, the *csh* command

```
setenv TEXINPUTS .:/usr/me/mylib:/usr/local/lib/tex82
```

or the *sh* command sequence

```
TEXINPUTS=.:/usr/me/mylib:/usr/stanford/lib/tex82/macros
export TEXINPUTS
```

would cause all invocations of tex and its derivatives to look for \input files first in the current directory, then in a hypothetical user's "mylib", and finally in the system library. Normally, the

user will place the command sequence which sets up the TEXINPUTS environment variable in the *.cshrc* or *.profile* file. The Environment section below lists the relevant environment variables, and their defaults.

The *e* response to TeX's error prompt causes the *vi* editor to start up at the current line of the current file. The environment variable, TEXEDIT, that can be used to change the editor used. It should contain a string with "%s" indicating where the filename goes and "%d" indicating where the decimal linenumber (if any) goes. For example, a TEXEDIT string for *emacs* might be set by:

setenv TEXEDIT "/usr/stanford/bin/emacs -ltex-start -estartline %d %s"

(where *tex-start* is installed in the emacs loadpath, as it is here).

A convenient file in the library is null.tex, containing nothing. When tex can't find a file it thinks you want to input, it keeps asking you for another file name; responding 'null' gets you out of the loop if you don't want to input anything.

Two other TeX programs, *initex* and *virtex*, can be used to create fast-loading customized versions of TeX. The *initex* program is used to create a *format (.fmt)* file that permits fast loading of fonts and macro packages. After processing the fonts and definitions desired, a \dump command will create the format file. The format file is used by *virtex*. It needs to be given a format file name as the first thing it reads. A format file name is preceded by an &, which needs to be escaped with \ if given on the command line. So, for instance, one could create a file myfmt.fmt using initex, and then set up a cshell alias with
alias mytex "virtex \&myfmt"
to allow the use of "mytex paper".

ENVIRONMENT

TEXINPUTS
: Search path for \input and \openin files. It should be colon-separated, and start with ".". Default: .:/usr/stanford/lib/tex82/macros

TEXFONTS
: Search path for font metric files. Default: /usr/stanford/lib/tex82/fonts

TEXFORMATS
: Search path for format files. Default: /usr/stanford/lib/tex82/macros

TEXPOOL
: Search path for TeX strings. Default: /usr/stanford/lib/tex82

TEXEDIT
: Command template for switching to editor. Default: "/usr/ucb/vi +%d %s"

FILES

/usr/stanford/lib/tex82/macros/plain.tex	Default macros and fonts.
/usr/stanford/lib/tex82/plain.fmt	Format file for the above.
/usr/stanford/lib/tex82/fonts/*	Font metric files.
/usr/stanford/lib/tex82/tex.pool	Message strings for *TeX*.

SEE ALSO

Donald E. Knuth, *The TeXbook*
Michael Spivak, *The Joy of TeX*
Leslie Lamport, *The LaTeX Document Preparation System*
TUGBOAT Volumes 1 and 2.
latex(1)
dvip(1)
There is a public mailing list, su-unix-tex, at Diablo, for dissemination of news about TeX, LaTeX,

library files, etc. See the manual page for mailer to find out how to get onto the mailing list.

TRIVIA

TeX, pronounced properly, rhymes with "blecchhh." Note that the proper spelling in typewriter-like output is "TeX" and not "TEX" or "tex."

BUGS

Maybe there should be character other than & to specify format files, since if you forget the \ on the command line, it doesn't do what you want! Also, there is no way to read a TeX input file with no filename extension.

AUTHORS

TeX was designed by Donald E. Knuth, who implemented it using his WEB system for Pascal programs. Installed at Stanford by Howard Trickey, October 11, 1982. Subsequently, incorporated changes based on a port by Pavel Curtis at Cornell.

NAME

latex – TeX with a macro package preloaded

SYNOPSIS

latex [first line]

DESCRIPTION

LaTeX is a set of TeX macros that provides the user with a complete document-preparation system. LaTeX was inspired by Scribe, but makes no effort to emulate it. In addition to the capabilities of PLAIN TeX, LaTeX provides many features, including the following:

- Automatic generation of section numbers and table of contents.
- Various forms of list-making commands, like enumerated lists.
- Commands to generate references to page and section numbers from symbolic labels.
- Automatic numbering of and symbolic referencing to bibliography citations. (Under construction is an auxiliary program to get bibliography entries from a central bibliographic database using the citations in the text.)
- Floating of figures and tables using a very sophisticated placement algorithm.
- Commands for drawing pictures with lines and arrows (horizontal, vertical and slanted), circles and quarter circles.
- Very convenient commands for generating arrays and tabular layout of text.
- Commands that allow you to run only part of your document through TeX with all the page and section numbering coming out right.

According to LaTeX's designer, Leslie Lamport, LaTeX is not simply a collection of nifty macros. It is an integrated SYSTEM for producing documents. It provides the user with a coherent model --- a much simpler and more coherent model than TeX presents --- which behaves consistently. (For example, enumerated lists do the right thing when nested within one another, or when they contain or appear inside other commands.)

The following "document styles" are currently installed: article (short documents), report (longer ones, with chapters), suthesis (Stanford PhD theses), and letter (for letters). Document style options are 11pt and 12pt (to change the "normal" point side), and twocolumn. The "normal" page layout style described in the LaTeX manual is available, but it wastes a lot of space on Dover output. The use of the "fullpage" page layout style is recommended.

There is also a program called *slitex* for making slides. Run latex and input "slides" to get information about how to use it.

More information on LateX can be found in the user's manual, "The LaTeX Document Preparation System." Also consult lerrata.tex (obtained by running "latex lerrata.tex") for revisions to the current manual. The manual entry for *tex*(1) should also be consulted for general information on running LaTeX.

SEE ALSO

Leslie Lamport, *The LaTeX Document Preparation System.*
tex(1)
dvip(1)

AUTHORS

LaTeX was designed and implemented by Leslie Lamport. TeX was designed by Donald E. Knuth, who implemented it using his WEB system for Pascal programs. Installed at Stanford by Howard Trickey.

NAME

tex, latex, initex, virtex – text formatting and typesetting

SYNOPSIS

tex [first line]

latex [first line]

initex [first line]

virtex [first line]

DESCRIPTION

TeX formats interspersed text and commands and outputs a typesetter independent file (called *DVI* which is short for *DeVice Independent*). The TeX language is described in the publication *The TeXbook* by Donald E. Knuth. Old manuals *(TeX and METAFONT: New Directions in Typesetting)* should be put away where they will never be seen, since there are literally hundereds of differences from the old TeX78 system that it describes.

Any arguments given on the command line to the TeX programs are passed to them as the first input line. As described in the TeXbook, that line should begin with a file name or a \controlsequence. The normal usage is to say "**tex** *paper*" to start processing *paper.tex*. The name "paper" will be the "jobname", and is used in forming output file names. Directory paths are stripped from "paper" before forming the jobname, so all output will be in the current directory. If TeX doesn't get a file name in the first line, the jobname is "texput". The default '.tex' extension can be overridden by specifying an extension explicitly.

If there is no paper.tex in the current directory, TeX will look through a search path of directories to try to find it. The standard library on the default search path has the basic format package, plain.tex, described in the TeXbook, as well as several others. Note that it is hardly ever necessary to \input plain, since the *tex* program has preloaded it. This means that all of the control sequences discussed in the TeXbook are known to TeX. Another TeX processor in common use is *latex*. It is like tex in all ways except that it has the format package *latex.tex* preloaded instead of plain.tex. This is a Scribe-inspired document preparation system being developed by Leslie Lamport. See the manual page *latex*(1) for more information.

The output DVI file is written on *name.dvi* where *name* is the jobname. A log of error messages goes into *name.log*.

DVI files should be printed on the Dover with *dvip* (1). Note that there have been incompatible changes in the DVI format between TeX78 and TeX, so programs used to print TeX78 output will not work for TeX.

A note about fonts: the TeXbook says that Plain TeX uses the basic fonts cmr10, cmbx10, etc. In fact it uses amr10, ambx10, etc., and whenever the TeXbook implies that you should use a font name beginning with cm, you should substitute one beginning with am. (Cm stands for "Computer Modern", which hasn't been designed yet. Am stands for "Almost Computer Modern", the interim font family. A further source of confusion is that there actually are some cm fonts installed, but these are for use by the obsolete TeX78 and are slightly incompatible with TeX.)

There are some environment variables that can be used to set up directory paths to search when TeX opens a file for input. For example, the *csh* command

```
setenv TEXINPUTS .:/usr/me/mylib:/usr/local/lib/tex82
```

or the *sh* command sequence

```
TEXINPUTS=.:/usr/me/mylib:/usr/stanford/lib/tex82/macros
export TEXINPUTS
```

would cause all invocations of tex and its derivatives to look for \input files first in the current directory, then in a hypothetical user's "mylib", and finally in the system library. Normally, the

user will place the command sequence which sets up the TEXINPUTS environment variable in the *.cshrc* or *.profile* file. The Environment section below lists the relevant environment variables, and their defaults.

The *e* response to TeX's error prompt causes the *vi* editor to start up at the current line of the current file. The environment variable, TEXEDIT, that can be used to change the editor used. It should contain a string with "%s" indicating where the filename goes and "%d" indicating where the decimal linenumber (if any) goes. For example, a TEXEDIT string for *emacs* might be set by:

setenv TEXEDIT "/usr/stanford/bin/emacs -ltex-start -estartline %d %s"

(where *tex-start* is installed in the emacs loadpath, as it is here).

A convenient file in the library is null.tex, containing nothing. When tex can't find a file it thinks you want to input, it keeps asking you for another file name; responding 'null' gets you out of the loop if you don't want to input anything.

Two other TeX programs, *initex* and *virtex*, can be used to create fast-loading customized versions of TeX. The *initex* program is used to create a *format (.fmt)* file that permits fast loading of fonts and macro packages. After processing the fonts and definitions desired, a \dump command will create the format file. The format file is used by *virtex*. It needs to be given a format file name as the first thing it reads. A format file name is preceded by an &, which needs to be escaped with \ if given on the command line. So, for instance, one could create a file myfmt.fmt using initex, and then set up a cshell alias with
alias mytex "virtex \&myfmt"
to allow the use of "mytex paper".

ENVIRONMENT

TEXINPUTS
: Search path for \input and \openin files. It should be colon-separated, and start with ".". Default: .:/usr/stanford/lib/tex82/macros

TEXFONTS
: Search path for font metric files. Default: /usr/stanford/lib/tex82/fonts

TEXFORMATS
: Search path for format files. Default: /usr/stanford/lib/tex82/macros

TEXPOOL
: Search path for TeX strings. Default: /usr/stanford/lib/tex82

TEXEDIT
: Command template for switching to editor. Default: "/usr/ucb/vi +%d %s"

FILES

/usr/stanford/lib/tex82/macros/plain.tex	Default macros and fonts.
/usr/stanford/lib/tex82/macros/plain.fmt	Format file for the above.
/usr/stanford/lib/tex82/fonts/*	Font metric files.
/usr/stanford/lib/tex82/tex.pool	Message strings for *TeX*.

SEE ALSO

Donald E. Knuth, *The TeXbook*
Michael Spivak, *The Joy of TeX*
Leslie Lamport, *The LaTeX Document Preparation System*
TUGBOAT Volumes 1 and 2.
latex(1)
dvip(1)
There is a public mailing list, su-unix-tex, at Diablo, for dissemination of news about TeX, LaTeX,

library files, etc. See the manual page for mailer to find out how to get onto the mailing list.

TRIVIA

TeX, pronounced properly, rhymes with "blecchhh." Note that the proper spelling in typewriter-like output is "TeX" and not "TEX" or "tex."

BUGS

Maybe there should be character other than & to specify format files, since if you forget the \ on the command line, it doesn't do what you want! Also, there is no way to read a TeX input file with no filename extension.

AUTHORS

TeX was designed by Donald E. Knuth, who implemented it using his WEB system for Pascal programs. Installed at Stanford by Howard Trickey, October 11, 1982. Subsequently, incorporated changes based on a port by Pavel Curtis at Cornell.

NAME

tftp – trivial file transfer program

SYNOPSIS

tftp [host]

DESCRIPTION

Tftp is the user interface to the Internet Trivial File Transfer Protocol. The program allows a user to transfer file to and from a remote network site. (Most users will prefer to use *ftp*(1C), a more advanced file transfer protocol; however, some hosts support only tftp.)

TFTP COMMANDS

Once *tftp* is running, it recognizes the following commands:

connect *host-name [port]*

Set the host (and optionally, port) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not actually maintain connections between transfers; thus, the "connect" command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the *connect* command; the remote host can be specified as part of the *get* or *put* commands.

mode *transfer-mode*

Set the mode for transfers. *Transfer-mode* may be one of *ascii*, *binary*, or *mail*.

put *file ... destination*

Stores a file or set of files to the specified *destination*. *Destination* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers.

get *source ... file*

Gets a file or set of files from the specified *sources*. *source* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter form is used, the last hostname specified becomes the default for future transfers.

quit Exits from *tftp*.

verbose Toggles the "verbose" flag on and off.

trace Toggles the "trace" flag on and off.

status Displays the current status. Useful for seeing what default settings are.

rexmt *retransmission-timeout*

Set the per-packet retransmission timeout, in seconds.

timeout *total-transmission-timeout*

Set the total transmission timeout, in seconds.

? [*command-name ...*]

Gives one-line summaries of the specified commands. With no arguments, lists the possible commands.

COMMAND LINE OPTIONS

The remote host with which *tftp* is to communicate may be specified on the command line. If this is done, *tftp* will do an implicit *connect* command for this host name.

BUGS

Lots.

This manual page was created by reading the code. It is probably wrong.
The "verbose" flag apparently has no effect.

NAME

time — time a command

SYNOPSIS

time command

DESCRIPTION

The given command is executed; after it is complete, *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.

On a PDP-11, the execution time can depend on what kind of memory the program happens to land in; the user time in MOS is often half what it is in core.

The times are printed on the diagnostic output stream.

Time is built in to *csh*(1), using a different output format.

BUGS

Elapsed time is accurate to the second, while the CPU times are measured to the 100th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

Time is a built-in command to *csh*(1), with a much different syntax. This command is available as "/bin/time" to *csh* users.

NAME

timecheck – checks and sets Pup network time

SYNOPSIS

timecheck [(reset | confirm) [threshold in seconds]]

DESCRIPTION

Timecheck is a general purpose utility program that is used to maintain and report upon the local and network-wide idea of what time it is. Invoked without any arguments, it reports the time of day as indicated by the local clock, and by a network time server. It indicates which host on the network responded first as a timeserver.

If a keyword argument is given (actually, only the first letter is significant), the action taken varies quite a bit:

reset – means that if the local time is not within *threshold* seconds of the network time, the local time should be reset to the network time. If the local time is within bounds, no message is given; otherwise, both clocks are reported, and a message is given to indicate that the time has been changed. Note that only the super-user can reset the local time-of-day.

confirm – has the same action as reset, except that it pauses before reseting the time to get confirmation from standard input. The confirmation should be a string beginning with the letter 'y', or any other string to indicate 'no'.

The default threshold is 300 seconds (5 minutes). If a negative threshold is given, then the time will always be reset to the network server if one exists.

This program is normally run from /etc/rc, as

timecheck reset -1

to set the local time to the network time.

AUTHOR

Jeffrey Mogul

SEE ALSO

date(1)

DIAGNOSTICS

Should be self-explanatory. Several message can be generated because the net is down.

BUGS

The protocol used to communicate time between hosts has no mechanism for determining if the time is in fact correct.

The worst case that could happen is that a timeserver gets a drastically wrong time, and all other servers then follow suit. The only way to fix this is to kill off all of the servers on the net, set some machine's local clock to a reasonable time, and start a timeserver. Then, the other machines can restart their timeservers.

A more likely problem is that the time will tend to drift, with each host having a slightly different idea of the time.

NAME

tip, cu − connect to a remote system

SYNOPSIS

tip [−**v**] [−*speed*] system-name
tip [−**v**] [−*speed*] phone-number
cu phone-number [−**t**] [−**s** *speed*] [**-a** *acu*] [−**l** *line*] [−**#**]

DESCRIPTION

Tip and *cu* establish a full-duplex connection to another machine, giving the appearance of being logged in directly on the remote cpu. It goes without saying that you must have a login on the machine (or equivalent) to which you wish to connect. The preferred interface is *tip*. The *cu* interface is included for those people attached to the "call UNIX" command of version 7. This manual page describes only *tip*.

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well). A tilde ('~') appearing as the first character of a line is an escape signal; the following are recognized:

~^D ~. Drop the connection and exit (you may still be logged in on the remote machine).

~c [*name*] Change directory to name (no argument implies change to your home directory).

~! Escape to a shell (exiting the shell will return you to tip).

~> Copy file from local to remote. *Tip* prompts for the name of a local file to transmit.

~< Copy file from remote to local. *Tip* prompts first for the name of the file to be sent, then for a command to be executed on the remote machine.

~p *from* [*to*]
Send a file to a remote UNIX host. The put command causes the remote UNIX system to run the command string "cat > 'to'", while *tip* sends it the "from" file. If the "to" file isn't specified the "from" file name is used. This command is actually a UNIX specific version of the "~>" command.

~t *from* [*to*]
Take a file from a remote UNIX host. As in the put command the "to" file defaults to the "from" file name if it isn't specified. The remote host executes the command string "cat 'from';echo ^A" to send the file to *tip*.

~| Pipe the output from a remote command to a local UNIX process. The command string sent to the local UNIX system is processed by the shell.

~# Send a BREAK to the remote system. For systems which don't support the necessary *ioctl* call the break is simulated by a sequence of line speed changes and DEL characters.

~s Set a variable (see the discussion below).

~^Z Stop *tip* (only available with job control).

~? Get a summary of the tilde escapes

Tip uses the file /etc/remote to find how to reach a particular system and to find out how it should operate while talking to the system; refer to *remote*(5) for a full description. Each system has a default baud rate with which to establish a connection. If this value is not suitable, the baud rate to be used may be specified on the command line, e.g. "tip -300 mds".

When *tip* establishes a connection it sends out a connection message to the remote system; the default value, if any, is defined in /etc/remote.

When *tip* prompts for an argument (e.g. during setup of a file transfer) the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt, or an interrupt, will abort the dialogue and return you to the remote machine.

Tip guards against multiple users connecting to a remote system by opening modems and terminal lines with exclusive access, and by honoring the locking protocol used by *uucp*(1C).

During file transfers *tip* provides a running count of the number of lines transferred. When using the ~> and ~< commands, the "eofread" and "eofwrite" variables are used to recognize end-of-file when reading, and specify end-of-file when writing (see below). File transfers normally depend on tandem mode for flow control. If the remote system does not support tandem mode, "echocheck" may be set to indicate *tip* should synchronize with the remote system on the echo of each transmitted character.

When *tip* must dial a phone number to connect to a system it will print various messages indicating its actions. *Tip* supports the DEC DN-11 and Racal-Vadic 831 auto-call-units; the DEC DF02 and DF03, Ventel 212+, Racal-Vadic 3451, and Bizcomp 1031 and 1032 integral call unit/modems.

VARIABLES

Tip maintains a set of *variables* which control its operation. Some of these variable are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the "s" escape. The syntax for variables is patterned after *vi*(1) and *Mail*(1). Supplying "all" as an argument to the set command displays all variables readable by the user. Alternatively, the user may request display of a particular variable by attaching a '?' to the end. For example "escape?" displays the current escape character.

Variables are numeric, string, character, or boolean values. Boolean variables are set merely by specifying their name; they may be reset by prepending a '!' to the name. Other variable types are set by concatenating an '=' and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables. Variables may be initialized at run time by placing set commands (without the "~s" prefix in a file *.tiprc* in one's home directory). The −v option causes *tip* to display the sets as they are made. Certain common variables have abbreviations. The following is a list of common variables, their abbreviations, and their default values.

beautify
(bool) Discard unprintable characters when a session is being scripted; abbreviated *be*.

baudrate
(num) The baud rate at which the connection was established; abbreviated *ba*.

dialtimeout
(num) When dialing a phone number, the time (in seconds) to wait for a connection to be established; abbreviated *dial*.

echocheck
(bool) Synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted; default is *off*.

eofread
(str) The set of characters which signify and end-of-tranmission during a ~< file transfer command; abbreviated *eofr*.

eofwrite
(str) The string sent to indicate end-of-transmission during a ~> file transfer command; abbreviated *eofw*.

eol
(str) The set of characters which indicate an end-of-line. *Tip* will recognize escape

characters only after an end-of-line.

escape
(char) The command prefix (escape) character; abbreviated *es*; default value is '~'.

exceptions
(str) The set of characters which should not be discarded due to the beautification switch; abbreviated *ex*; default value is "\t\n\f\b".

force
(char) The character used to force literal data transmission; abbreviated *fo*; default value is '^P'.

framesize
(num) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated *fr*.

host
(str) The name of the host to which you are connected; abbreviated *ho*.

prompt
(char) The character which indicates and end-of-line on the remote host; abbreviated *pr*; default value is '\n'. This value is used to synchronize during data transfers. The count of lines transferred during a file transfer command is based on recipt of this character.

raise
(bool) Upper case mapping mode; abbreviated *ra*; default value is *off*. When this mode is enabled, all lower case letters will be mapped to upper case by *tip* for transmission to the remote machine.

raisechar
(char) The input character used to toggle upper case mapping mode; abbreviated *rc*; default value is '^A'.

record
(str) The name of the file in which a session script is recorded; abbreviated *rec*; default value is "tip.record".

script
(bool) Session scripting mode; abbreviated *sc*; default is *off*. When *script* is *true*, *tip* will record everything transmitted by the remote machine in the script record file specified in *record*. If the *beautify* switch is on, only printable ASCII characters will be included in the script file (those characters betwee 040 and 0177). The variable *exceptions* is used to indicate characters which are an exception to the normal beautification rules.

tabexpand
(bool) Expand tabs to spaces during file transfers; abbreviated *tab*; default value is *false*. Each tab is expanded to 8 spaces.

verbose
(bool) Verbose mode; abbreviated *verb*; default is *true*. When verbose mode is enabled, *tip* prints messages while dialing, shows the current number of lines transferred during a file transfer operations, and more.

SHELL
(str) The name of the shell to use for the ~! command; default value is "/bin/sh", or taken from the environment.

HOME
(str) The home directory to use for the ~c command; default value is taken from the

environment.

FILES

/etc/remote	global system descriptions
/etc/phones	global phone number data base
${REMOTE}	private system descriptions
${PHONES}	private phone numbers
~/.tiprc	initialization file.
/usr/spool/uucp/LCK..*	lock file to avoid conflicts with *uucp*

DIAGNOSTICS

Diagnostics are, hopefully, self explanatory.

SEE ALSO

remote(5), phones(5)

BUGS

The full set of variables is undocumented and should, probably, be paired down.

NAME

tk − paginator for the Tektronix 4014

SYNOPSIS

tk [**−t**] [*−N*] [**−p***L*] [file]

DESCRIPTION

The output of *tk* is intended for a Tektronix 4014 terminal. *Tk* arranges for 66 lines to fit on the screen, divides the screen into *N* columns, and contributes an eight space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. Teletype Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page *tk* waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command *!command* will send the *command* to the shell.

The command line options are:

−t Don't wait between pages; for directing output into a file.

−N Divide the screen into *N* columns and wait after the last column.

−p*L* Set page length to *L* lines.

SEE ALSO

pr(1)

NAME

touch − update date last modified of a file

SYNOPSIS

touch [−c] [−f] file ...

DESCRIPTION

Touch attempts to set the modified date of each *file*. If a *file* exists, this is done by reading a character from the file and writing it back. If a *file* does not exist, an attempt will be made to create it unless the **−c** option is specified. The **−f** option will attempt to force the touch in spite of read and write permissions on a *file*.

SEE ALSO

utimes(2)

NAME

tp — manipulate tape archive

SYNOPSIS

tp [key] [name ...]

DESCRIPTION

Tp saves and restores files on DECtape or magtape. Its actions are controlled by the *key* argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. 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 The named files are written on the tape. If files with the same names already exist, they are replaced. 'Same' is determined by string comparison, so './abc' can never be the same as '/usr/dmr/abc' even if '/usr/dmr' is the current directory. If no file argument is given, '.' is the default.

u updates the tape. **u** is like **r**, but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. **u** is the default command if none is given.

d deletes the named files from the tape. At least one name argument must be given. This function is not permitted on magtapes.

x extracts the named files from the tape to the file system. The owner and mode are restored. If no file argument is given, the entire contents of the tape are extracted.

t lists the names of the specified files. If no file argument is given, the entire contents of the tape is listed.

The following characters may be used in addition to the letter which selects the function desired.

m Specifies magtape as opposed to DECtape.

0,...,7 This modifier selects the drive on which the tape is mounted. For DECtape, **x** is default; for magtape '0' is the default.

v Normally *tp* 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.

c means a fresh dump is being created; the tape directory is cleared before beginning. Usable only with **r** and **u**. This option is assumed with magtape since it is impossible to selectively overwrite magtape.

i Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

f Use the first named file, rather than a tape, as the archive. This option currently acts like **m**; *i.e.* **r** implies **c**, and neither **d** nor **u** are permitted.

w causes *tp* to pause before treating each file, type the indicative letter and the file name (as with **v**) and await the user's response. Response **y** means 'yes', so the file is treated. Null response means 'no', and the file does not take part in whatever is being done. Response **x** means 'exit'; the *tp* command terminates immediately. In the **x** function, files previously asked about have been extracted already. With **r**, **u**, and **d** no change has been made to the tape.

FILES

/dev/tap?
/dev/rmt?

SEE ALSO

ar(1), tar(1)

DIAGNOSTICS

Several; the non-obvious one is 'Phase error', which means the file changed after it was selected for dumping but before it was dumped.

BUGS

A single file with several links to it is treated like several files.

Binary-coded control information makes magnetic tapes written by *tp* difficult to carry to other machines; *tar*(1) avoids the problem.

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*. When *string2* is short it is padded to the length of *string1* by duplicating its last character. Any combination of the options **−cds** may be used: **−c** complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal; **−d** deletes all input characters in *string1;* **−s** squeezes all strings of repeated output characters that are in *string2* to single characters.

In either string the notation *a−b* means a range of characters from *a* to *b* in increasing ASCII order. The character '\' followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A '\' followed by any other character stands for that character.

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 second string is quoted to protect '\' from the Shell. 012 is the ASCII code for newline.

```
tr −cs A−Za−z '\012' <file1 >file2
```

SEE ALSO

ed(1), ascii(7), expand(1)

BUGS

Won't handle ASCII NUL in *string1* or *string2;* always deletes NUL from input.

NAME

trman − translate version 6 manual macros to version 7 macros

SYNOPSIS

trman [file]

DESCRIPTION

Trman reads the input file, which should be nroff/troff input and attempts to translate the version 6 manual sections therein to version 7 format. It is largely successful, but seems to have trouble with indented paragraphs and complicated font control. You should expect to have to fix up long sections by hand somewhat.

SEE ALSO

man(7)

BUGS

NAME

troff, nroff − text formatting and typesetting

SYNOPSIS

troff [option] ... [file] ...

nroff [option] ... [file] ...

DESCRIPTION

Troff formats text in the named *files* for printing on a Graphic Systems C/A/T phototypesetter; *nroff* is used for for typewriter-like devices. Their capabilities are described in the *Nroff/Troff user's manual.*

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (−) is taken to be a file name corresponding to the standard input. The options, which may appear in any order so long as they appear before the files, are:

−o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N*−*M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end.

−n*N* Number first generated page *N*.

−s*N* Stop every *N* pages. *Nroff* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline. *Troff* will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed.

−m*name* Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

−r*aN* Set register *a* (one-character) to *N*.

−i Read standard input after the input files are exhausted.

−q Invoke the simultaneous input-output mode of the **rd** request.

Troff only

−t Direct output to the standard output instead of the phototypesetter.

−f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

−w Wait until phototypesetter is available, if currently busy.

−b Report whether the phototypesetter is busy or available. No text processing is done.

−a Send a printable ASCII approximation of the results to the standard output.

−pN Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.

If the file */usr/adm/tracct* is writable, *troff* keeps phototypesetter accounting records there. The integrity of that file may be secured by making *troff* a 'set user-id' program.

FILES

/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*
/usr/lib/font/*	font width tables for *troff*
/dev/cat	phototypesetter
/usr/adm/tracct	accounting statistics for /dev/cat

SEE ALSO

J. F. Ossanna, *Nroff/Troff user's manual*
B. W. Kernighan, *A TROFF Tutorial*
eqn(1), tbl(1), ms(7), me(7), man(7), col(1)

NAME

true, false — provide truth values

SYNOPSIS

true

false

DESCRIPTION

True and *false* are usually used in a Bourne shell script. They test for the appropriate status "true" or "false" before running (or failing to run) a list of commands.

EXAMPLE

```
while true
do
    command list
done
```

SEE ALSO

csh(1), sh(1), false(1)

DIAGNOSTICS

True has exit status zero.

NAME

tset − terminal dependent initialization

SYNOPSIS

tset [options] [**−m** [ident][test baudrate]:type] ... [type]

reset ...

DESCRIPTION

Tset sets up your terminal when you first log in to a UNIX system. It does terminal dependent processing such as setting erase and kill characters, setting or resetting delays, sending any sequences needed to properly initialized the terminal, and the like. It first determines the *type* of terminal involved, and then does necessary initializations and mode settings. The type of terminal attached to each UNIX port is specified in the */etc/ttytype* database. Type names for terminals may be found in the *termcap*(5) database. If a port is not wired permanently to a specific terminal (not hardwired) it will be given an appropriate generic identifier such as *dialup*.

In the case where no arguments are specified, *tset* simply reads the terminal type out of the environment variable TERM and re-initializes the terminal. The rest of this manual concerns itself with mode and environment initialization, typically done once at login, and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script (*.profile* for *sh*(1) users or *.login* for *csh*(1) users) it is desirable to give information about the type of terminal you will usually use on ports which are not hardwired. These ports are identified in */etc/ttytype* as *dialup* or *plugboard* or *arpanet*, etc. To specify what terminal type you usually use on these ports, the **−m** (map) option flag is followed by the appropriate port type identifier, an optional baud rate specification, and the terminal type. (The effect is to "map" from some conditions to a terminal type, that is, to tell *tset* "If I'm on this kind of port, guess that I'm on that kind of terminal".) If more than one mapping is specified, the first applicable mapping prevails. A missing port type identifier matches all identifiers. Any of the alternate generic names given in *termcap* may be used for the identifier.

A *baudrate* is specified as with *stty*(1), and is compared with the speed of the diagnostic output (which should be the control terminal). The baud rate *test* may be any combination of: **>**, **@**, **<**, and **!**; **@** means "at" and **!** inverts the sense of the test. To avoid problems with metacharacters, it is best to place the entire argument to **−m** within "'" characters; users of *csh*(1) must also put a "\" before any "!" used here.

Thus

```
tset −m 'dialup>300:adm3a' -m dialup:dw2 -m 'plugboard:?adm3a'
```

causes the terminal type to be set to an *adm3a* if the port in use is a dialup at a speed greater than 300 baud; to a *dw2* if the port is (otherwise) a dialup (i.e. at 300 baud or less). **(NOTE:** the examples given here appear to take up more than one line, for text processing reasons. When you type in real *tset* commands, you must enter them entirely on one line.) If the *type* finally determined by *tset* begins with a question mark, the user is asked if s/he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. Thus, in the above case, the user will be queried on a plugboard port as to whether they are actually using an *adm3a*.

If no mapping applies and a final *type* option, not preceded by a **−m**, is given on the command line then that type is used; otherwise the identifier found in the */etc/ttytype* database will be taken to be the terminal type. This should always be the case for hardwired ports.

It is usually desirable to return the terminal type, as finally determined by *tset*, and information about the terminal's capabilities to a shell's environment. This can be done using the **−** option; using the Bourne shell, *sh*(1):

export TERM; TERM=`tset − *options...*`

or using the C shell, *csh*(1):

setenv TERM `tset - *options...*`

With *csh* it is convenient to make an alias in your .cshrc:

alias tset 'setenv TERM `tset − \!*`'

Either of these aliases allow the command

tset 2621

to be invoked at any time from your login csh. **Note to Bourne Shell users:** It is **not** possible to get this aliasing effect with a shell script, because shell scripts cannot set the environment of their parent. (If a process could set its parent's environment, none of this nonsense would be necessary in the first place.)

These commands cause *tset* to place the name of your terminal in the variable TERM in the environment; see *environ*(7).

Once the terminal type is known, *tset* engages in terminal driver mode setting. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the line-kill (full line erase)) characters, and setting special character delays. Tab and newline expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character ('#' on standard systems), the erase character is changed to BACKSPACE (Control-H).

The options are:

−e*c* set the erase character to be the named character *c* on all terminals, the default being the backspace character on the terminal, usually ^H. The character *c* can either be typed directly, or entered using the hat notation used here.

−k*c* is similar to **−e** but for the line kill character rather than the erase character; *c* defaults to ^X (for purely historical reasons). The kill characters is left alone if **−k** is not specified. The hat notation can also be used for this option.

− The name of the terminal finally decided upon is output on the standard output. This is intended to be captured by the shell and placed in the environment variable TERM.

−n On systems with the Berkeley 4BSD tty driver, specifies that the new tty driver modes should be initialized for this terminal. For a CRT, the CRTERASE and CRTKILL modes are set only if the baud rate is 1200 or greater. See tty(4) for more detail.

−I suppresses transmitting terminal initialization strings.

−Q suppresses printing the "Erase set to" and "Kill set to" messages.

If **tset** is invoked as **reset**, it will set cooked and echo modes, turn off cbreak and raw modes, turn on newline translation, and restore special characters to a sensible state before any terminal dependent processing is done. Any special character that is found to be NULL or "−1" is reset to its default value.

This is most useful after a program dies leaving a terminal in a funny state. You may have to type "<LF>reset<LF>" to get it to work since <CR> may not work in this state. Often none of this will echo.

EXAMPLES

These examples all assume the Bourne shell and use the - option. If you use *csh*, use one of the variations described above. Note that a typical use of *tset* in a .profile or .login will also use the **−e** and **−k** options, and often the **−n** or **−Q** options as well. These options have not

been included here to keep the examples small. (**NOTE:** some of the examples given here appear to take up more than one line, for text processing reasons. When you type in real *tset* commands, you must enter them entirely on one line.)

At the moment, you are on a 2621. This is suitable for typing by hand but not for a .profile, unless you are *always* on a 2621.

```
export TERM; TERM=`tset - 2621`
```

You have an h19 at home which you dial up on, but your office terminal is hardwired and known in /etc/ttytype.

```
export TERM; TERM=`tset - -m dialup:h19`
```

You have a switch which connects everything to everything, making it nearly impossible to key on what port you are coming in on. You use a vt100 in your office at 9600 baud, and dial up to switch ports at 1200 baud from home on a 2621. Sometimes you use someone elses terminal at work, so you want it to ask you to make sure what terminal type you have at high speeds, but at 1200 baud you are always on a 2621. Note the placement of the question mark, and the quotes to protect the greater than and question mark from interpretation by the shell.

```
export TERM; TERM=`tset - -m 'switch>1200:?vt100' -m 'switch<=1200:2621'
```

All of the above entries will fall back on the terminal type specified in */etc/ttytype* if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals. Your most common terminal is an adm3a. It always asks you what kind of terminal you are on, defaulting to adm3a.

```
export TERM; TERM=`tset - ?adm3a`
```

If the file */etc/ttytype* is not properly installed and you want to key entirely on the baud rate, the following can be used:

```
export TERM; TERM=`tset - -m '>1200:vt100' 2621`
```

Here is a fancy example to illustrate the power of *tset* and to hopelessly confuse anyone who has made it this far. You dial up at 1200 baud or less on a concept100, sometimes over switch ports and sometimes over regular dialups. You use various terminals at speeds higher than 1200 over switch ports, most often the terminal in your office, which is a vt100. However, sometimes you log in from the university you used to go to, over the ARPANET; in this case you are on an ALTO emulating a dm2500. You also often log in on various hardwired ports, such as the console, all of which are properly entered in */etc/ttytype*. You want your erase character set to control H, your kill character set to control U, and don't want *tset* to print the "Erase set to Backspace, Kill set to Control U" message.

```
export TERM; TERM=`tset -e -k^U -Q - -m 'switch<=1200:concept100' -m
'switch:?vt100' -m dialup:concept100 -m arpanet:dm2500`
```

FILES

/etc/ttytype	port name to terminal type mapping database
/etc/termcap	terminal capability database

SEE ALSO

csh(1), sh(1), stty(1), ttytype(5), termcap(5), environ(7)

AUTHORS

Eric Allman
David Wasley
Mark Horton

BUGS

The *tset* command is one of the first commands a user must master when getting started on a UNIX system. Unfortunately, it is one of the most complex, largely because of the extra effort the user must go through to get the environment of the login shell set. Something needs to be done to make all this simpler, either the *login*(1) program should do this stuff, or a default shell alias should be made, or a way to set the environment of the parent should exist.

NAME

tsort — topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort 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.

SEE ALSO

lorder(1)

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

ttime -- measure terminal output rate

SYNOPSIS

ttime [–] <ttyname> [. . .]

DESCRIPTION

Ttime is used to measure the rate at which characters can be printed on a terminal, or set of terminals. It is useful in debugging or tuning programs for network terminal access.

Ttime is invoked with a list of terminal names; it prints characters on each of them until stopped with an "interrupt" signal (e.g., CTRL/C from the controlling terminal.) Unless the "–" flag is given, *ttime* also runs a CPU-bound process.

Normal use of *ttime* is to invoke it as
ttime – /dev/tty
After waiting a few minutes, interrupt the program and it will tell you how many characters were printed, per second.

AUTHOR

Peter Eichenberger

BUGS

It's not clear what the cpu-bound process is meant to do.

CURRENT RECORD FOR HIGHEST OUTPUT RATE

VGTS+iptn running on Sun-1/upgrade via 10mb ethernet to a VAX-11/750 running 4.2BSD: **3877 characters per second.**

NAME

tty − get terminal name

SYNOPSIS

tty [-s]

DESCRIPTION

Tty prints the pathname of the user's terminal unless the −s (silent) is given. In either case, the exit value is zero if the standard input is a terminal and one if it is not.

DIAGNOSTICS

'not a tty' if the standard input file is not a terminal.

NAME

ul − do underlining

SYNOPSIS

ul [**−i**] [**−t** *terminal*] [*name* ...]

DESCRIPTION

Ul reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable TERM. The **−t** option overrides the terminal kind specified in the environment. The file */etc/termcap* is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, *ul* degenerates to *cat*(1). If the terminal cannot underline, underlining is ignored.

The **−i** option causes *ul* to indicate underlining onto by a separate line containing appropriate dashes '−'; this is useful when you want to look at the underlining which is present in an *nroff* output stream on a crt-terminal.

SEE ALSO

man(1), nroff(1), colcrt(1)

AUTHOR

Mark Horton wrote *ul*. The **−i** option was originally a option of the editor *ex*(1), then an *iul* command.

BUGS

Nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

NAME

undump – convert a core dump to an executable a.out file

SYNOPSIS

undump new-a.out-file [old-a.out-file] [core-file]

DESCRIPTION

Undump takes a core dump file and the executable "a.out" file which caused it and produces a new executable file with all static variables initialised to the values they held at the time of the core dump. It is primarily useful for programs which take a long time to initialise themselves, e.g. Emacs. The idea is to go through all of the initialisations and then create a core dump (e.g. with the abort() call). One then uses undump to make a new executable file with all of it done. This usually implies the use of a global flag variable which says whether or not initialisation has been done.

Undump's arguments, old-a.out-file and core-file, default to "a.out" and "core", respectively.

A few things to keep in mind about undump:

It doesn't preserve open files.

The program will be re-entered at the beginning of main(), not at the point where the core dump occurred.

BUGS

Probably should have an option to not require old-a.out-file if the core came from a 407 file.

NAME

unifdef – remove ifdef'ed lines

SYNOPSIS

unifdef [**−t** **−l** **−c** **−D***sym* **−U***sym* **−id***sym* **−iu***sym*] ... [file]

DESCRIPTION

Unifdef is useful for removing ifdef'ed lines from a file while otherwise leaving the file alone. *Unifdef* is like a stripped-down C preprocessor: it is smart enough to deal with the nested ifdefs, comments, single and double quotes of C syntax so that it can do its job, but it doesn't do any including or interpretation of macros. Neither does it strip out comments, though it recognizes and ignores them. You specify which symbols you want defined **−D***sym* or undefined **−U***sym* and the lines inside those ifdefs will be copied to the output or removed as appropriate. The ifdef, ifndef, else, and endif lines associated with *sym* will also be removed. Ifdefs involving symbols you don't specify are untouched and copied out along with their associated ifdef, else, and endif lines. If an ifdef X occurs nested inside another ifdef X, then the inside ifdef is treated as if it were an unrecognized symbol. If the same symbol appears in more than one argument, only the first occurrence is significant.

The **−l** option causes *unifdef* to replace removed lines with blank lines instead of deleting them.

If you use ifdefs to delimit non-C lines, such as comments or code which is under construction, then you must tell *unifdef* which symbols are used for that purpose so that it won't try to parse for quotes and comments in those ifdef'ed lines. You specify that you want the lines inside certain ifdefs to be ignored but copied out with **−id***sym* and **−iu***sym* similar to **−D***sym* and **−U***sym* above.

If you want to use *unifdef* for plain text (not C code), use the **−t** option. This makes *unifdef* refrain from attempting to recognize comments and single and double quotes.

Unifdef copies its output to *stdout* and will take its input from *stdin* if no *file* argument is given. If the **−c** argument is specified, then the operation of *unifdef* is complemented, i.e. the lines that would have been removed or blanked are retained and vice versa.

SEE ALSO

diff(1)

DIAGNOSTICS

Premature EOF, inappropriate else or endif.

Exit status is 0 if output is exact copy of input, 1 if not, 2 if trouble.

BUGS

Does not know how to deal with *cpp* consructs such as

```
#if     defined(X) || defined(Y)
```

AUTHOR

Dave Yost

NAME

unpcnt – remove lines beginning with % from a file

SYNOPSIS

unpcnt

DESCRIPTION

Unpcnt reads the standard input and copies it to the standard output, deleting lines that begin with the '%' character. This is useful for taking programs that have been run through VERCH before feeding them to a compiler.

AUTHOR

Jeffrey Mogul

BUGS

Lines longer than 1000 characters are (silently) botched.

NAME

unscribe -- remove Scribe constructs

SYNOPSIS

unscribe file ...

DESCRIPTION

Unscribe is the Scribe equivalent of *deroff(1)* with the **-w option.** It reads each file in sequence, removes all numbers, punctuation, and Scribe commands, and writes the remainder on the standard output, one word per line. Single-character words are omitted. If no input file is given, *unscribe* reads from the standard input file.

The local version of *spell(1)* uses *unscribe* in place of *deroff* when given the **-s** flag. This is the primary use of *unscribe.*

SEE ALSO

scribe(1), deroff(1), spell(1)

AUTHOR

Tim Mann

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. 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

sort(1), comm(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
* *2.54000e+00*
/ *3.93701e−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 pounds force/in2
You want: atm
* *1.02069e+00*
/ *9.79730e−01*

Units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade 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 a unit of mass. Compound names are run together, e.g. 'lightyear'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. Currency is denoted 'belgiumfranc', 'britainpound', ...

For a complete list of units, 'cat /usr/lib/units'.

FILES

/usr/lib/units

BUGS

Don't base your financial plans on the currency conversions.

NAME

uptime — show how long system has been up

SYNOPSIS

uptime

DESCRIPTION

Uptime prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a *w*(1) command.

FILES

/vmunix system name list

SEE ALSO

w(1)

NAME

users — compact list of users who are on the system

SYNOPSIS

users

DESCRIPTION

Users lists the login names of the users currently on the system in a compact, one-line format.

FILES

/etc/utmp

SEE ALSO

who(1)

NAME

uucp, uulog — unix to unix copy

SYNOPSIS

uucp [option] ... source-file ... destination-file

uulog [option] ...

DESCRIPTION

Uucp copies files named by the source-file arguments to the destination-file argument. A file name may be a path name on your machine, or may have the form

system-name!pathname

where 'system-name' is taken from a list of system names which *uucp* knows about. Shell metacharacters ?*[] appearing in the pathname part will be expanded on the appropriate system.

Pathnames may be one of

(1) a full pathname;

(2) a pathname preceded by ~*user*; where *user* is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current 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)).

The following options are interpreted by *uucp*.

-d Make all necessary directories for the file copy.

-c Use the source file when copying out rather than copying the file to the spool directory.

-m Send mail to the requester when the copy is complete.

Uulog maintains a summary log of *uucp* and *uux*(1C) transactions in the file '/usr/spool/uucp/LOGFILE' by gathering information from partial log files named '/usr/spool/uucp/LOG.*.?'. It removes the partial log files.

The options cause *uulog* to print logging information:

-s*sys* Print information about work involving system *sys*.

-u*user*
Print information about work done for the specified *user*.

FILES

/usr/spool/uucp - spool directory
/usr/lib/uucp/* - other data and program files

SEE ALSO

uux(1C), mail(1)
D. A. Nowitz, *Uucp Implementation Description*

WARNING

The domain of remotely accessible files can (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.

BUGS

All files received by *uucp* will be owned by uucp.
The −m option will only work sending files or receiving a single file. (Receiving multiple files specified by special shell characters ?*[] will not activate the −m option.)

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 (or other) mail. This combination can be used over indirect mail links even when *uusend*(1C) is not available.

Uuencode takes the named source file (default 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 recreation 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* or by making *rmail* be a link to *Mail* instead of *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.

SEE ALSO

uuencode(5), uusend(1C), uucp(1C), uux(1C), mail(1)

AUTHOR

Mark Horton

BUGS

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking *uudecode* (often *uucp)* must have write permission on the specified file.

NAME

uusend – send a file to a remote host

SYNOPSIS

uusend [**–m** mode] sourcefile sys1!sys2!..!remotefile

DESCRIPTION

Uusend sends a file to a given location on a remote system. The system need not be directly connected to the local system, but a chain of *uucp*(1C) links needs to connect the two systems.

If the **–m** option is specified, the mode of the file on the remote end will be taken from the octal number given. Otherwise, the mode of the input file will be used.

The sourcefile can be "–", meaning to use the standard input. Both of these options are primarily intended for internal use of uusend.

The remotefile can include the ~userid syntax.

DIAGNOSTICS

If anything goes wrong any further away than the first system down the line, you will never hear about it.

SEE ALSO

uux(1C), uucp(1C), uuencode(1)

AUTHOR

Mark Horton

BUGS

This command shouldn't exist, since *uucp* should handle it.

All systems along the line must have the *uusend* command available and allow remote execution of it.

Some uucp systems have a bug where binary files cannot be the input to a uux command. If this bug exists in any system along the line, the file will show up severly munged.

NAME

uux – unix to unix command execution

SYNOPSIS

uux [–] command-string

DESCRIPTION

Uux will gather 0 or more files from various systems, execute a command on a specified system and send 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 file names may be prefixed by system-name!. A null system-name is interpreted as the local system.

File names may be one of

(1) a full pathname;

(2) a pathname preceded by ~*xxx*; where *xxx* is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

The '–' option will cause the standard input to the *uux* command to be the standard input to the command-string.

For example, the command

uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !fi.diff"

will get the f1 files from the usg and pwba machines, execute a *diff* command and put the results in f1.diff in the local directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire command-string, or quoting the special characters as individual arguments.

FILES

/usr/spool/uucp spool directory
/usr/lib/uucp/* other data and programs

SEE ALSO

uucp(1C)
D. A. Nowitz, *Uucp Implementation Description*

WARNING

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an incoming request from *uux*. Typically, a restricted site will permit little other than the receipt of mail via *uux*.

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.
There is no notification of denial of execution on the remote machine.

NAME

verch — version changing program for Pascal sources

SYNOPSIS

verch Infile [Outfile] [-flag1+] [-flag2-]...
verch Infile [Outfile] [-strip]

DESCRIPTION

VERCH is a program that can be used to change back and forth between different versions of a Pascal program. To convert from one version to another, you run the program through VERCH, and it comments out blocks of code that are not relevent to the current version, and uncomments blocks that are, according to commands which are contained in special-format comments in the code itself. Thus both the input and output to VERCH are compilable Pascal programs.

In the Pascal source code, version command line begins with the characters "(*%" followed by one of the following commands: SETT, SETF, SET?, IFT, IFF, ELSE, ENDC, followed by a flag name. Spaces and tabs can be used freely, and comments can be added after the last part of the command.

(*% Sett S1 *)	set flag "S1" to "true"
(*% Setf S1 *)	set flag "S1" to "false"
(*% Set? S1 *)	get value of flag "S1" from user
(*% Ift S1 *)	if flag "S1" true then don't comment out this block
(*% Iff S1 *)	if flag "S1" false then don't comment out this block
(*% Else *)	begin reverse block
(*% Endc *)	end of block
(*% Delete *)	Delete blocks instead of commenting them out (Also deletes all command lines.)
(*% Strip *)	Uncomment everything, no matter what the value of the flags
(*% Nopage *)	Delete page marks
(*% Kchars *)	To provide for Dec-10 compiler kludge, use "%" and ":" as the commenting-out characters instead of "{" and "}"

If "T", "F", or "?" appear after a SETT, SETF, or SET? command, then the version changer will replace it with the current value of the flag. Thus you can tell when you look at a program what version is currently active. For instance, we can tell by the following line that the program hasn't been run through the version changer yet, and thus the version "S1" is still inactive:
(*% SETT S1 F *)

The flags may also be set directly from the command line, as shown above. The flags are set as switches, with *flag1-* indicating that flag1 should be set to False and *flag2+* indicating that flag2 should be set to True. The command *strip* (see below) may also be set from the command line. If VERCH encounters the use of a flag which has not been set, it will query the user for the value to insert.

A full-blown example: suppose we wanted to change the Vax version of a program to the S-1 version of the program. Here is the program before and after running it through VERCH:

BEFORE: **********

(*%SETF Vax T *)

```
(*%IFT Vax *)
  program PTRANS (INPUT,OUTPUT);
(*%ELSE*)
{}
{  program PTRANS;}
{}
(*%ENDC*)
  begin
  end.

AFTER: ************************************************************

(*%SETF Vax F *)
(*%IFT Vax *)
{  program PTRANS (INPUT,OUTPUT);}
(*%ELSE*)
  program PTRANS;
(*%ENDC*)
  begin
  end.

************************************************************************
```

Note that if we had used "SET?" instead of "SETF", the version changer would have asked us what we wanted the flag set to.
Other details:

"AND" and "OR" can be used to concatenate IFF and IFT commands. For instance, after reading the following command, the version changer would not comment out the block that followed if either flag were true:
(*% IFT S1 OR IFT VAX *)

Blocks may be nested. The AND function is performed on nested blocks. In other words, if any of the blocks containing a statement are "false" blocks, the statement is commented out.

Only the first 16 characters of a flag are significant.

The original idea for the version-changer came from Dick Sites.

This program depends on having at least two kinds of comments characters which are distinguished, e.g. (* *), and { }, so that comments may be nested at least one level deep. This is contrary to the Pascal Standard, which mandates that a right curly bracket be treated identically with a "*)", but most compilers support it.

Verch has a limit on the number of switches allowed on the command line (currently set to ten). Thus, there is a maximum on the number of flags that may be set from the command line.

If no output file name is given, Verch will name the output file by appending the extension **new** ®to the input file name.

SEE ALSO

unpent(1)

DIAGNOSTICS

Verch now writes diagnostics to the standard Pascal output, so they are user-visible.

NAME

vfontinfo − inspect and print out information about UNIX fonts

SYNOPSIS

vfontinfo [**−v**] fontname [characters]

DESCRIPTION

Vfontinfo allows you to examine a font in the UNIX format. It prints out all the information in the font header and information about every non-null (width > 0) glyph. This can be used to make sure the font is consistent with the format.

The *fontname* argument is the name of the font you wish to inspect. It writes to standard output. If it can't find the file in your working directory, it looks in */usr/lib/vfont* (the place most of the fonts are kept).

The *characters*, if given, specify certain characters to show. If omitted, the entire font is shown.

If the **−v** (verbose) flag is used, the bits of the glyph itself are shown as an array of X's and spaces, in addition to the header information.

SEE ALSO

vpr(1), vfont(5)
The Berkeley Font Catalog

AUTHORS

Mark Horton
Andy Hertzfeld

NAME

vgrind − grind nice listings of programs

SYNOPSIS

vgrind [**−f**] [**−**] [**−t**] [**−n**] [**−x**] [**−W**] [**−sn**] [**−h** header] [**−d** file] [**−l***language*] name ...

DESCRIPTION

Vgrind formats the program sources which are arguments in a nice style using *troff*(1) Comments are placed in italics, keywords in bold face, and the name of the current function is listed down the margin of each page as it is encountered.

Vgrind runs in two basic modes, filter mode or regular mode. In filter mode *vgrind* acts as a filter in a manner similar to *tbl*(1). The standard input is passed directly to the standard output except for lines bracketed by the *troff-like* macros:

.vS - starts processing

.vE - ends processing

These lines are formatted as described above. The output from this filter can be passed to *troff* for output. There need be no particular ordering with *eqn*(1) or *tbl*(1).

In regular mode *vgrind* accepts input files, processes them, and passes them to *troff*(1) for output.

In both modes *vgrind* passes any lines beginning with a decimal point without conversion.

The options are:

−f forces filter mode

− forces input to be taken from standard input (default if **−f** is specified)

−t similar to the same option in *troff* causing formatted text to go to the standard output

−n forces no keyword bolding

−x outputs the index file in a "pretty" format. The index file itself is produced whenever *vgrind* is run with a file called *index* in the current directory. The index of function definitions can then be run off by giving *vgrind* the **−x** option and the file *index* as argument.

−W forces output to the (wide) Versatec printer rather than the (narrow) Varian

−s specifies a point size to use on output (exactly the same as the argument of a .ps)

−h specifies a particular header to put on every output page (default is the file name)

−d specifies an alternate language definitions file (default is /usr/lib/vgrindefs)

−l specifies the language to use. Currently known are PASCAL (**−lp**), MODEL (**−lm**),C (**−lc** or the default), CSH (**−lcsh**), SHELL (**−lsh**), RATFOR (**−lr**), and ICON (**−ll**).

FILES

index	file where source for index is created
/usr/lib/tmac/tmac.vgrind	macro package
/usr/lib/vfontedpr	preprocessor
/usr/lib/vgrindefs	language descriptions

AUTHOR

Dave Presotto & William Joy

SEE ALSO

vlp(1), vtroff(1), vgrindefs(5)

BUGS

Vfontedpr assumes that a certain programming style is followed:

For **C** – function names can be preceded on a line only by spaces, tabs, or an asterisk. The parenthesized arguments must also be on the same line.

For **PASCAL** – function names need to appear on the same line as the keywords *function* or *procedure.*

For **MODEL** – function names need to appear on the same line as the keywords *is beginproc.*

If these conventions are not followed, the indexing and marginal function name comment mechanisms will fail.

More generally, arbitrary formatting styles for programs mostly look bad. The use of spaces to align source code fails miserably; if you plan to *vgrind* your program you should use tabs. This is somewhat inevitable since the font used by *vgrind* is variable width.

The mechanism of ctags in recognizing functions should be used here.

NAME

vi − screen oriented (visual) display editor based on ex

SYNOPSIS

vi [**−t** tag] [**−r**] [**+***command*] [**−l**] [**−w***n*] name ...

DESCRIPTION

Vi (visual) is a display oriented text editor based on *ex*(1). *Ex* and *vi* run the same code; it is possible to get to the command mode of *ex* from within *vi* and vice-versa.

The *Vi Quick Reference* card and the *Introduction to Display Editing with Vi* provide full details on using *vi.*

FILES

See *ex*(1).

SEE ALSO

ex (1), edit (1), "Vi Quick Reference" card, "An Introduction to Display Editing with Vi".

AUTHOR

William Joy
Mark Horton added macros to *visual* mode and is maintaining version 3

BUGS

Software tabs using ^T work only immediately after the *autoindent.*

Left and right shifts on intelligent terminals don't make use of insert and delete character operations in the terminal.

The *wrapmargin* option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line won't be broken.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a **:** escape. To use these on a **:global** you must **Q** to *ex* command mode, execute them, and then reenter the screen editor with *vi* or *open.*

NAME

vlp − Format Lisp programs to be printed with nroff, vtroff, or troff

SYNOPSIS

vlp [**−p** *pointsize*] [**−d**] [**−f**] [**−l**] [**−v**] [**−T** *title1*] file1 [**−T** *title2*] file2 ...

DESCRIPTION

Vlp formats the named files so that they can be run through nroff, vtroff, or troff to produce listings that line-up and are attractive. The first non-blank character of each line is lined-up vertically, as in the source file. Comments (text beginning with a semicolon) are printed in italics. Each function's name is printed in bold face next to the function. This format makes Lisp code look attractive when it is printed with a variable width font.

Normally, *vlp* works as a filter and sends its output to the standard output. However, the **−v** switch pipes the output directly to vtroff. If no files are specified, then *vlp* reads from the standard input.

The following options are available:

−p The **−p** switch changes the size of the text from its default value of 8 points to one of 6, 8, 10, or 12 points. Once set, the point size is used for all subsequent files. This point size does not apply to embedded text (see *−f* below).

−d The **−d** switch puts *vlp* into debugging mode.

−f *Vlp* has a filtered mode in which all lines are passed unmodified, except those lines between the directives **.Ls** and **.Le**. This mode can be used to format Lisp code that is embedded in a document. The directive **.Ls** takes an optional argument that gives the point size for the embedded code. If not size is specified, the size of the surrounding text is used.

−l The **−l** switch prevents *vlp* from placing labels next to functions. This switch is useful for embedded Lisp code, where the labels would be distracting.

−v This switch cause *vlp* to send its output to vtroff rather than the standard output.

−T A title to be printed on each page may be specified by using the **−T** switch. The **−T** switch applies only to the next file name given. Titles are not printed for embedded text (see **−f**, above). This switch may not be used if *vlp* is reading from the standard input.

FILES

/usr/lib/vlpmacs troff/nroff macros

AUTHOR

Originally written by John K. Foderaro, with additional changes by Kevin Layer and James Larus.

SEE ALSO

vgrind(1), lisp(1)

BUGS

vlp transforms \ into \\ so that it will be printed out. Hence, troff commands cannot be embedded in Lisp code.

NAME

vmstat − report virtual memory statistics

SYNOPSIS

vmstat [**−fs**] [interval [count]]

DESCRIPTION

Vmstat delves into the system and normally reports certain statistics kept about process, virtual memory, disk, trap and cpu activity. If given a **−f** argument, it instead reports on the number of *forks* and *vforks* since system startup and the number of pages of virtual memory involved in each kind of fork. If given a **−s** argument, it instead prints the contents of the *sum* structure, giving the total number of several kinds of paging related events which have occurred since boot.

If none of these options are given, *vmstat* will report in the first line a summary of the virtual memory activity since the system has been booted. If *interval* is specified, then successive lines are summaries over the last *interval* seconds. "vmstat 5" will print what the system is doing every five seconds; this is a good choice of printing interval since this is how often some of the statistics are sampled in the system; others vary every second, running the output for a while will make it apparent which are recomputed every second. If a *count* is given, the statistics are repeated *count* times. The format fields are:

Procs: information about numbers of processes in various states.

r	in run queue
b	blocked for resources (i/o, paging, etc.)
w	runnable or short sleeper (< 20 secs) but swapped

Memory: information about the usage of virtual and real memory. Virtual pages are considered active if they belong to processes which are running or have run in the last 20 seconds. A "page" here is 1024 bytes.

avm	active virtual pages
fre	size of the free list

Page: information about page faults and paging activity. These are averaged each five seconds, and given in units per second.

re	page reclaims (simulating reference bits)
pi	pages paged in
po	pages paged out
fr	pages freed per second
de	anticipated short term memory shortfall
sr	pages scanned by clock algorithm, per-second

up/hp/rk: Disk operations per second (this field is system dependent). Typically paging will be split across several of the available drives. The number under each of these is the unit number.

Faults: trap/interrupt rate averages per second over last 5 seconds.

in	(non clock) device interrupts per second
sy	system calls per second
cs	cpu context switch rate (switches/sec)

Cpu: breakdown of percentage usage of CPU time

us	user time for normal and low priority processes
sy	system time
id	cpu idle

FILES

/dev/kmem, /vmunix

SEE ALSO

The sections starting with "Interpreting system activity" in *Installing and Operating 4.2bsd.*

AUTHORS

William Joy and Ozalp Babaoglu

BUGS

There should be a screen oriented program which combines *vmstat* and *ps*(1) in real time as well as reporting on other system activity.

NAME

vnews – read news articles

SYNOPSIS

vnews [**–a** *date*] [**–n** *newsgroups*] [**–t** *titles*] [**–rxu**]

vnews –s

DESCRIPTION

Vnews is a program for reading USENET news. It is based on readnews but has a CRT oriented interface. The list of available commands is quite similar, although since vnews is a "visual" interface, most vnews commands do not have to be terminated by a newline.

Vnews uses the first 22 lines of the screen to display the current article. Line 23 is the secondary prompt line, and is used to input string arguments to commands. Line 24 contains several fields. The first field is the prompt field. If vnews is at the end of an article, the prompt is "next?"; otherwise the prompt is "more?". The second field is the newsgroup field, which displays the current newsgroup, the number of the current article, and the number of the last article in the newsgroup. The third field contains the current time, and the last field contains the word "mail" if you have mail. When you receive new mail, the bell on the terminal is rung and the word mail appears in capital letters for 30 seconds.

The **–r** without any arguments prints unread articles. flag causes the articles to be printed in reverse order. The **–u** flag causes the **.newsrc** file to be updated every 5 minutes, in case of an unreliable system. (Note that if the newsrc file is updated, the **x** command will not restore it to its original contents.)

The following flags determine the selection of articles.

–n *newsgroups*
: Select all articles that belong to *newsgroups*.

–t *titles*
: Select all articles whose titles contain one of the strings specified by *titles*.

–a [*date*]
: Select all articles that were posted past the given *date* (in *getdate*(3) format).

–x
: Ignore *.newsrc* file. That is, select articles that have already been read as well as new ones.

vnews maintains a *.newsrc* file in the your home directory that specifies all news articles already read. It is updated at the end of each reading session in which the **–x** option wasn't specified. If the environment variable NEWSRC is present, it should be the path name of a file to be used in place of .newsrc.

If you wish, an options line may be placed in your *.newsrc* file. This line starts with the word **options** (left justified) followed by the list of standard options just as they would be typed on the command line. Such a list may include: the **–n** flag along with a newsgroup list; and/or the **–r** or **–t** flag. Continuation lines are specified by following lines beginning with a space or tab character.

vnews –s will print the newsgroup subscription list.

ENVIRONMENT

Options can be specified in the **NEWSOPTS** environment parameter. Where conflicts exist, options on the command line take precedence, followed by the *.newsrc* **options** line, and lastly the **NEWSOPTS** parameter.

When the user uses the reply command, the environment parameter **MAILER** will be used to determine which mailer to use. The default is usually /bin/mail.

If the user so desires, he may specify a specific paging program for articles. The environment parameter **PAGER** should be set to the paging program. The name of the article is referenced with

a '%', as in the −c option. If no '%' is present, the article will be piped to the program. Paging may be disabled by setting **PAGER** to a null value.

If **EDITOR** is set, it will be used in place of the default editor on your system to edit replies and follow-ups.

If **NAME** is set, it will be used as your full name when posting news or submitting a follow-up. If it is not set, the name will be taken from the file **.name** in your home directory. If this file is not present, the name will be taken from /etc/passwd.

If **NEWSARCHIVE** is set, a copy of any articles you post or follow-up to, will be save in the specified file. If it is the null string, they will be copied in *author_copy* in your home directory.

If **NEWSBOX** is set, when you save or write a file, if the filename you specify does not begin with a "/", it will be prepended with NEWSBOX.

If **NEWSRC** is set, it will be used in place of the .newsrc file in your home directory.

If **ORGANIZATION** is set, it will be used as the name of your organization whenever you post an article. The default is compiled in and is usually correct. Typically, you would only use this if you were reading news at a site other than normal. (Or if you are trying to be cute.)

COMMANDS

Each vnews command may be preceded by a count. Some commands use the count; others ignore it. If count is omitted, it defaults to one. Some commands prompt for an argument on the second line from the bottom of the screen. Standard UNIX erase and kill processing is done on this argument. The argument is terminated by a return. An interrupt (delete or break) gets you out of any partially entered command.

In the following table, ↑B is used as a shorthand for Control-B.

Command	Meaning
CR	A carriage return prints more of the current article, or goes on to the next article if you are at the end of the current article. A SPACE is equivalent to CR.
↑B	Goes backwards count pages.
↑F	Goes forward count pages.
↑D	Go forwards half a page.
↑U	Go backwards half a page.
↑Z	Go forwards count lines.
↑E	Go backwards count lines.
↑L	Redraws the screen. ↑L may be typed at any time.
b	Back up one article in the current group.
c	Cancel the article. Only the author of the article or the super user can do this.
d	Read a digest. Breaks up a digest into separate articles and permits you to read and reply to each piece.
e	Erase. Forget that this article was read.
f	Submit a follow-up article. You will be placed in your EDITOR to compose the text of the follow-up.
h	Go back to the top of the article and display only the header.
l	Redisplays the article after you have sent a follow-up or reply.
n	No. Goes on to next article without printing current one. "." is equivalent to "n". This is

convenient if your terminal has a keypad.

p Gets you the parent article (the article that the current article is a follow-up to). This doesn't work if the current article was posted by A-news or notesfiles. To get back to from the parent article, use the - command. Unfortunately, if you use several p commands to trace the discussion back further, there is no command to get you back.

q Quit. The .newsrc file will be updated if −x was not on the command line.

r Reply. Reply to article's author via mail. You are placed in your EDITOR with a header specifying To, Subject, and References lines taken from the message. You may change or add headers, as appropriate. You add the text of the reply after the blank line, and then exit the editor. The resulting message is mailed to the author of the article.

s [*file*] Save. The article is appended to the named file. The default is "Articles". If the first character of the file name is '|', the rest of the file name is taken as the name of a program, which is executed with the text of the article as standard input. If the first character of the file name is '/', it is taken as a full path name of a file. If $NEWSBOX (in the environment) is set to a full path name, and the file contains no '/', the file is saved in $NEWSBOX. Otherwise, it is saved relative to $HOME.

ug Unsubscribe to the current group. This is a two character command to ensure that it is not typed accidentally and to leave room for other types of unsubscribes (e. g. unsubscribe to discussion).

v Print the current version of the news software.

w Is the same as "s", expect that the headers are not written out.

x Exit. Like quit except that .newsrc is not updated.

y Yes. Prints the current article and goes on to the next.

[*n*]A Go to article number *n* in the current newsgroup.

D Decrypts a joke. It only handles rot 13 jokes. The D command is a toggle; typing another D re- encrypts the joke.

H Print a very verbose header, containing all known information about the article.

K Kill (mark as read) the rest of the articles in the current group. This is useful if you can't keep up with the volume in the newsgroup, but don't want to unsubscribe.

N [*newsgroup*]
Go to the next newsgroup or named newsgroup.

[*n*]+ Skip n articles. The articles skipped are recorded as "unread" and will be offered to you again the next time you read news.

− Go back to last article. This is a toggle, typing it twice returns you to the original article.

< Prompts for an article ID or the rest of a message ID. It will display the article if it exists.

Report the name and size of the newsgroup.

? Print an short help message.

! Passes the rest of the command line to the shell. The environment variable $A is set to the name of the file containing the current article. If the last character of the command is a "&", then the "&" is deleted and the command is run in the background with stdin, stdout and stderr redirected to /dev/null. If the command is missing, the shell is invoked. Use the 1 command (or essentially any other command) to turn on the display after the program terminates.

EXAMPLES

vnews Read all unread articles using the *visual* interface. The *.newsrc* file is updated at the end of the session.

vnews −n all !fa.all −r
Read all unread articles except articles whose newsgroups begin with "fa." in reverse order. The *.newsrc* file is updated at the end of the session.

vnews −n all −a last thursday
Print every unread article since last Thursday. The *.newsrc* file is updated at the end of the session.

vnews −p > /dev/null &
Discard all unread news. This is useful after returning from a long trip.

FILES

/usr/spool/news/*newsgroup/number*	News articles
/usr/lib/news/active	Active newsgroups and numbers of articles
/usr/lib/news/vnews.help	Help file for *visual* interface
~/.newsrc	Options and list of previously read articles

SEE ALSO

checknews(1), readnews(1), recnews(8), news(5), newsrc(5)

NAME

vpr, vprm, vpq, vprint − raster printer/plotter spooler

SYNOPSIS

vpr [**−W**] [**−l**] [**−v**] [**−t** [**−1234** font]] [**−w**] [**−w***width*] [**−m**] [name ...]
vprm [id ...] [filename ...] [owner ...]
vpq
vprint [**−W**] file ...

DESCRIPTION

Vpr causes the named files to be queued for printing or typeset simulation on one of the available raster printer/plotters. If no files are named, the standard input is read. By default the input is assumed to be line printer-like text. For very wide plotters, the input is run through the filter */usr/lib/sidebyside* giving it an argument of **−w106** which arranges it four pages adjacent with 90 column lines (the rest is for the left margin). Since there are 8 lines per inch in the default printer font, *vpr* thus produces 86 lines per page (the top and bottom lines are left blank).

The following options are available:

−l Print the input in a more literal manner. Page breaks are not inserted, and most control characters (except format effectors: \n, \f, etc.) are printed (many control characters print special graphics not in the ASCII character set.) Tab and underline processing is still done. If this option is not given, control characters which are not format effectors are ignored, and page breaks are inserted after an appropriate number of lines have been printed on a page.

−W Queues files for printing on a wide output device, if available. Normally, files are queued for printing on a narrow output device.

−1234 Specifies a font to be mounted on font position *i*. The daemon will construct a *.railmag* file referencing */usr/lib/vfont/name.size*.

−m Report by *mail*(1) when printing is complete.

−w (Applicable only to wide output devices.) Do not run the input through sideby-side. Such processing has been done already, or full (440 character) printer width is desired.

−w*width* Use width *width* rather than 90 for *sidebyside*.

−v Use the filter */usr/lib/vrast* to convert the vectors to raster. The named files must be a parameter and vector file (in that order) created by *plot*(3X) routines.

−t Use the filter */usr/lib/vcat* to typeset the input on the printer/plotter. The input must have been generated by *troff*(1) run with the **−t** option. This is not normally run directly to wide output devices, since it is wasteful to run only one page across. The program *vtroff*(1) is normally used and arranges, using *vsort* for printing to occur four pages across, conserving paper.

Vprm removes entries from the raster device queues. The id, filename or owner should be that reported by *vpq*. All appropriate files will be removed. Both queues are always searched. The id of each file removed from the queue will be printed.

Vpq prints the queues. Each entry in the queue is printed showing the owner of the queue entry, an identification number, the size of the entry in characters, and the file which is to be printed. The *id* is useful for removing a specific entry from the printer queue using *vprm*

Vprint is a shell script which *pr's* a copy of each named file on one of the electrostatic printer/plotters. The files are normally printed on a narrow device; −W option causes them to be printed on a wide device.

FILES

/usr/spool/v?d/*	device spool areas
/usr/lib/v?d	daemons
/usr/lib/vpd	Versatec daemon
/usr/lib/vpf	filter for printer simulation
/usr/lib/*vcat	filter for typeset simulation
/usr/lib/vrast	filter for plot
/usr/lib/sidebyside	filter for wide output

SEE ALSO

troff(1), vfont(5), vp(4), pti(1), vtroff(1), plot(3X)

BUGS

The 1's (one's) and l's (lower-case el's) in a Benson-Varian's standard character set look very similar; caution is advised.

A versatec's hardware character set is rather ugly. *Vprint* should use one of the constant width fonts to produce prettier listings.

NAME

vtroff − troff to a raster plotter

SYNOPSIS

vtroff [**−w**] [**−F** majorfont] [**−123** minorfont] [**−l***length*] [**−x**] troff arguments

DESCRIPTION

Vtroff runs *troff*(1) sending its output through various programs to produce typeset output on a raster plotter such as a Benson-Varian or or a Versatec. The **−W** option specifies that a wide output device be used; the default is to use a narrow device. The **−l** (lower case l) option causes the output to be split onto successive pages every *length* inches rather than the default 11".

The default font is a Hershey font. If some other font is desired you can give a **−F** argument and then the font name. This will place normal, italic and bold versions of the font on positions 1, 2, and 3. To place a font only on a single position, you can give an argument of the form **−***n* and the minor font name. A **.r** will be added to the minor font name if needed. Thus "vtroff −ms paper" will set a paper in the Hershey font, while "vtroff −F nonie −ms paper" will set the paper in the (sans serif) nonie font. The **−x** option asks for exact simulation of photo-typesetter output. (I.e. using the width tables for the C.A.T. photo-typesetter)

FILES

/usr/lib/tmac/tmac.vcat	default font mounts and bug fixes
/usr/lib/fontinfo/*	fixes for other fonts
/usr/lib/vfont	directory containing fonts

SEE ALSO

troff(1), vfont(5), vpr(1)

BUGS

Since some macro packages work correctly only if the fonts named R, I, B, and S are mounted, and since the Versatec fonts have different widths for individual characters than the fonts found on the typesetter, the following dodge was necessary: If you don't use the ".fp" troff directive then you get the widths of the standard typesetter fonts suitable for shipping the output of troff over the network to the computer center A machine for phototypesetting. If, however, you remount the R, I, B and S fonts, then you get the width tables for the Versatec.

NAME

vwidth − make troff width table for a font

SYNOPSIS

vwidth fontfile pointsize > ft*xx*.c
cc -c ft*xx*.c mv ft*xx*.o /usr/lib/font/ft*xx*

DESCRIPTION

Vwidth translates from the width information stored in the vfont style format to the format expected by troff. Troff wants an object file in a.out(5) format. (This fact does not seem to be documented anywhere.) Troff should look directly in the font file but it doesn't.

Vwidth should be used after editing a font with *fed(1)*. It is not necessary to use vwidth unless you have made a change that would affect the width tables. Such changes include numerically editing the width field, adding a new character, and moving or copying a character to a new position. It is *not* always necessary to use vwidth if the physical width of the glyph (e.g. the number of columns in the bit matrix) has changed, but if it has changed much the logical width should probably be changed and vwidth run.

Vwidth produces a C program on its standard output. This program should be run through the C compiler and the object (that is, the .o file) saved. The resulting file should be placed in /usr/lib/font in the file ft*xx* where is a one or two letter code that is the logical (internal to troff) font name. This name can be found by looking in the file /usr/lib/fontinfo/*fname** where *fname* is the external name of the font.

SEE ALSO

fed(1), vfont(5), troff(1), vtroff(1)

BUGS

Produces the C file using obsolete syntax that the portable C compiler complains about.

NAME

w − who is on and what they are doing

SYNOPSIS

w [−h] [−s] [user]

DESCRIPTION

W prints a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields output are: the users login name, the name of the tty the user is on, the time of day the user logged on, the number of minutes since the user last typed anything, the CPU time used by all processes and their children on that terminal, the CPU time used by the currently active processes, the name and arguments of the current process.

The **−h** flag suppresses the heading. The **−s** flag asks for a short form of output. In the short form, the tty is abbreviated, the login time and cpu times are left off, as are the arguments to commands. **−l** gives the long output, which is the default.

If a *user* name is included, the output will be restricted to that user.

FILES

/etc/utmp
/dev/kmem
/dev/drum

SEE ALSO

who(1), finger(1), ps(1)

AUTHOR

Mark Horton

BUGS

The notion of the "current process" is muddy. The current algorithm is "the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal". This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. (In cases where no process can be found, *w* prints "−".)

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is "charged" with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

W does not know about the new conventions for detection of background jobs. It will sometimes find a background job instead of the right one.

NAME

wait — await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with & have completed, and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the Shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1)

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the Shell, and thus can't be waited for. (This bug does not apply to *csh*(1).)

NAME

wall – write to all users

SYNOPSIS

wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by 'Broadcast Message ...', to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

FILES

/dev/tty?
/etc/utmp

SEE ALSO

mesg(1), write(1)

DIAGNOSTICS

'Cannot send to ...' when the open on a user's tty file fails.

NAME

wc − word count

SYNOPSIS

wc [**−lwc**] [name ...]

DESCRIPTION

Wc counts lines, words and characters in the named files, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs or newlines.

If an argument beginning with one of "lwc" is present, the specified counts (lines, words, or characters) are selected by the letters **l**, **w**, or **c**. The default is **−lwc**.

BUGS

NAME

what − show what versions of object modules were used to construct a file

SYNOPSIS

what name ...

DESCRIPTION

What reads each file and searches for sequences of the form "@(#)" as inserted by the source code control system. It then prints the remainder of the string after this marker, up to a null character, newline, double quote, or ">" character.

BUGS

As SCCS is not licensed with UNIX/32V, this is a rewrite of the *what* command which is part of SCCS, and may not behave exactly the same as that command does.

NAME

whatis − describe what a command is

SYNOPSIS

whatis command ...

DESCRIPTION

Whatis looks up a given command and gives the header line from the manual section. You can then run the *man*(1) command to get more information. If the line starts 'name(section) ...' you can do 'man section name' to get the documentation for it. Try 'whatis ed' and then you should do 'man 1 ed' to get the manual.

Whatis is actually just the **−f** option to the *man*(1) command.

FILES

/usr/lib/whatis Data base

SEE ALSO

man(1), catman(8)

AUTHOR

William Joy

NAME

whereami – report name of terminal

SYNOPSIS

whereami [-]

DESCRIPTION

Whereami reports on the standard output the name of your terminal, and whether it is a local or network terminal. This is especially useful for network users, since there is no fixed assignment between physical terminals and terminal names.

If any argument is given, the program prints nothing unless the terminal is a network terminal. Thus, *whereami* is meant to be used in a .profile or .cshrc file, with an argument, so that if you log in over a network, you will be told where you are.

AUTHOR

Jeffrey Mogul

BUGS

The program actually tries to find the name of the terminal associated with stderr, so if stderr is not a tty, a facetious message is printed. This could be smarter, but why waste the effort?

The mapping from terminal name to "networkness" is wired into the code, and must be changed when the pty assignments are.

NAME

whereis − locate source, binary, and or manual for program

SYNOPSIS

whereis [−sbm] [−u] [−SBM dir ... −f] name ...

DESCRIPTION

Whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form ".ext", e.g. ".c". Prefixes of "s." resulting from use of source code control are also dealt with. *Whereis* then attempts to locate the desired program in a list of standard places. If any of the **−b, −s** or **−m** flags are given then *whereis* searches only for binaries, sources or manual sections respectively (or any two thereof). The **−u** flag may be used to search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus "whereis -m -u *" asks for those files in the current directory which have no documentation.

Finally, the **−B −M** and **−S** flags may be used to change or otherwise limit the places where *whereis* searches. The **−f** file flags is used to terminate the last such directory list and signal the start of file names.

EXAMPLE

The following finds all the files in /usr/bin which are not documented in /usr/man/man1 with source in /usr/src/cmd:

```
cd /usr/ucb
whereis −u −M /usr/man/man1 −S /usr/src/cmd −f *
```

FILES

/usr/src/*
/usr/{doc,man}/*
/lib, /etc, /usr/{lib,bin,ucb,old,new,local}

AUTHOR

William Joy

BUGS

Since the program uses *chdir*(2) to run faster, pathnames given with the **−M −S** and **−B** must be full; i.e. they must begin with a "/".

NAME

which — locate a program file including aliases and paths (*csh* only)

SYNOPSIS

which [name] ...

DESCRIPTION

Which takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are taken from the user's .cshrc file.

FILES

~/.cshrc source of aliases and path values

DIAGNOSTICS

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

BUGS

Must be executed by a csh, since only csh's know about aliases.

NAME

who − who is on the system

SYNOPSIS

who [who-file] [**am I**]

DESCRIPTION

Who, without an argument, lists the login name, terminal name, and login time for each current UNIX user.

Without an argument, *who* examines the */etc/utmp* file to obtain its information. If a file is given, that file is examined. Typically the given file will be /usr/adm/wtmp, which contains a record of all the logins since it was created. Then *who* lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with '/dev/' suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with 'x' in the place of the device name, and a fossil time indicative of when the system went down.

With two arguments, as in 'who am I' (and also 'who are you'), *who* tells who you are logged in as.

FILES

/etc/utmp

SEE ALSO

getuid(2), utmp(5)

NAME

whoami — print effective current user id

SYNOPSIS

whoami

DESCRIPTION

Whoami prints who you are. It works even if you are su'd, while 'who am i' does not since it uses /etc/utmp.

FILES

/etc/passwd Name data base

SEE ALSO

who (1)

NAME

whois – ask the ARPA Internet NIC about a user

SYNOPSIS

whois [–s host] [–p port] [–v] [–h] ident ...

DESCRIPTION

The *whois* program implements the user end of the ARPA Internet NICNAME/WHOIS protocol, as described in RFC812. To quote from that document:

> "The NICNAME/WHOIS Server is an NCP/TCP transaction based query/response server, running on the SRI-NIC machine, that provides netwide directory service to ARPANET users. It is one of a series of ARPANET/Internet name services maintained by the Network Information Center (NIC) at SRI International on behalf of the Defense Communications Agency (DCA). The server is accessible across the ARPANET from user programs running on local hosts, and it delivers the full name, U.S. mailing address, telephone number, and network mailbox for ARPANET users."

Invoked with a user's name as an argument (embedded spaces are allowed), *whois* asks the NICNAME server about the user, and prints the response. For example,

```
% whois dyer, mary
Connecting to server on sri-nic for "dyer, mary"... open.

Dyer, Mary K. (MARY)                    DYER@SRI-NIC
  SRI International
  Network Information Center
  Telecommunications Sciences Center
  333 Ravenswood Avenue
  Menlo Park, California 94025
  Phone: (415) 859-4775
```

If you use the '–h' switch, *whois* will ask the NICNAME server to print a help text. The '–v' switch tells *whois* to disclose the precise transaction occuring. The '–s host' switch may be used to specify a host name other than **sri–nic**, similarly, the '–p port' switch may be used to specify a port number other than the port for the NICNAME server. This port number may either be a TCP service name (see *services* (5)) or a decimal number. These last two switches are useful only for debugging things.

FILES

/etc/hosts

SEE ALSO

finger (1), fing(l)

DIAGNOSTICS

Obvious.

BUGS

None known.

NAME

write − write to another user

SYNOPSIS

write user [ttyname]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message

Message from yoursystem!yourname yourttyname...

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. 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 *ttyname* argument may be used to indicate the appropriate terminal name.

Permission to write may be denied or granted by use of the *mesg* command. At the outset writing is allowed. Certain commands, in particular *nroff* and *pr*(1) disallow messages in order to prevent messy output.

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 him to write back before starting to send. Each party should end each message with a distinctive signal—**(o)** for 'over' is conventional—that the other may reply. **(oo)** for 'over and out' is suggested when conversation is about to be terminated.

FILES

/etc/utmp	to find user
/bin/sh	to execute '!'

SEE ALSO

mesg(1), who(1), mail(1)

NAME

xsend, xget, enroll – secret mail

SYNOPSIS

xsend person
xget
enroll

DESCRIPTION

These commands implement a secure communication channel; it is like *mail*(1), but no one can read the messages except the intended recipient. The method embodies a public-key cryptosystem using knapsacks.

To receive messages, use *enroll*; it asks you for a password that you must subsequently quote in order to receive secret mail.

To receive secret mail, use *xget*. It asks for your password, then gives you the messages.

To send secret mail, use *xsend* in the same manner as the ordinary mail command. (However, it will accept only one target). A message announcing the receipt of secret mail is also sent by ordinary mail.

FILES

/usr/spool/secretmail/*.key: keys
/usr/spool/secretmail/*.[0-9]: messages

SEE ALSO

mail (1)

BUGS

It should be integrated with ordinary mail. The announcement of secret mail makes traffic analysis possible.

NAME

xstr − extract strings from C programs to implement shared strings

SYNOPSIS

xstr [**−c**] [−] [file]

DESCRIPTION

Xstr maintains a file *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

xstr −c name

will extract the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An appropriate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file *x.c*, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file *xs.c* declaring the common *xstr* space can be created by a command of the form

xstr

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

Xstr can also be used on a single file. A command

xstr name

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. *Xstr* reads from its standard input when the argument '−' is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc −E name.c | xstr −c −
cc −c x.c
mv x.o name.o
```

Xstr does not touch the file *strings* unless new items are added, thus *make* can avoid remaking *xs.o* unless truly necessary.

FILES

strings	Data base of strings
x.c	Massaged C source
xs.c	C source for definition of array 'xstr'
/tmp/xs*	Temp file when 'xstr name' doesn't touch *strings*

SEE ALSO

mkstr(1)

AUTHOR

William Joy

BUGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings will be placed in the data base, when just placing the longer one there will do.

NAME

yacc − yet another compiler-compiler

SYNOPSIS

yacc [**−vd**] grammar

DESCRIPTION

Yacc 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.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(1)
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.
YACC − Yet Another Compiler Compiler by S. C. Johnson.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard 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.

BUGS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.

NAME

yapp — yet another pretty printer

SYNOPSIS

yapp [-t] [-*.ext*] files ...

DESCRIPTION

Yapp is yet another attempt at producing aesthetically pleasing program listings, which is a contradiction in terms.

It does not try to re-format your program; all it does it change fonts. Reserved words, comments, and other things each print in a different font; a fixed-width font is used for leading white space so as to preserve indentation.

Normally, *yapp* guesses the language in which your program is written by looking at the extension part of its filename; you can force *yapp* to use reserved words pertinent to a different extension by using the −. flag, where *ext* is the extension you wish to "force."

Currently, *yapp* knows about C, Ada, Bliss, and Pascal reserved words.

The −t option leaves the intermediate file around in an unnamable place. Otherwise, it gets printed and deleted.

FILES

/usr/stanford/lib/press/*_words reserved words files
/tmp/yapp* intermediate file

SEE ALSO

vgrind(1), cz(1)

DIAGNOSTICS

Ha.

AUTHOR

James Gosling

BUGS

This program reflects the taste (or lack thereof) of its author. The other (although numerous) bugs are irrelevant in the face of this.

NAME

yes – be repetitively affirmative

SYNOPSIS

yes [*expletive*]

DESCRIPTION

Yes repeatedly outputs "y", or if *expletive* is given, that is output repeatedly. Termination is by rubout.