CTIX™ OPERATING SYSTEM MANUAL

Version C
Volume 2

This manual was prepared on a Convergent Technologies S/320 Computer System and was printed on an Apple LaserWriter II Laser Printer.

Second Edition (November 1989) 09-02263-01

TABLE OF CONTENTS: VOLUME 2

HOW TO USE THIS MANUAL

This second edition of the *CTIX Operating System Manual, Version C*, describes the commands, system calls, libraries, data files, and device interfaces that make up the CTIX Operating System for S/Series Computer Systems. This manual should always be your starting point when you need to find the documentation for a CTIX feature with which you are unfamiliar.

The manual consists of a large number of short entries, sometimes called "the *man* pages," after the command that accesses the entries when they are kept online. Each entry briefly documents some feature of CTIX. Some features require longer documentation than an entry in this manual; such features have an entry that outlines the feature and cross-references the manual that documents the feature fully. Entries that do not refer to other manuals are self-contained and are the final word on the features they describe.

Organization of the manual. The entries are organized into seven sections in four volumes:

Volumes 1 and 2:

1. Commands and Application Programs.

Volume 3:

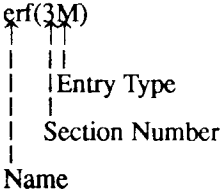
2. System Calls.
3. Subroutines and Libraries.

Volume 4:

4. File Formats.
5. Miscellaneous Facilities.
6. Games.
7. Special Files.

Within each section, entries are alphabetical by title, except for an *intro* entry at the beginning of each section.

Entry Title Conventions. An entry title looks like this example:



Name is the name of the entry. *Section Number* indicates the section that contains the entry. In this case, the entry is in Section 3, which is in Volume 2. *Entry Type* appears only on entries that belong to special categories; refer to the section's *intro* entry for an explanation. In this case, a reference to *intro*(3) would tell you that *erf*(3M) describes functions from the Math Library, which the C compiler does not load by default.

Finding the entry you need. To find out which entry you need, refer to the following guides:

- The Permuted Index. This indexes each significant word in each entry's description. It is useful when you have only a general notion what you're looking for. It is also useful when you know the name of the command or function you are interested in, but there is no entry by that name.
- The Table of Contents. This is a simple list of entries, by section, together with the entry descriptions. Volumes 1 and 2 have Tables of Contents for Section 1. Volume 3 has a Table of Contents for Sections 2 and 3. Volume 4 has a Table of Contents for Sections 4 through 7.
- The Table of Related Entries. For Volume 1 only. A table of entries organized so that related entries are grouped together.

Section organization. Each section begins with an *intro* entry, which provides important general information for that section.

Section 1, Commands and Application Programs, describes programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory **/bin** (for **bin**ary programs). Some programs also reside in **/usr/bin**, to save space in **/bin**. These directories are searched automatically by the command interpreter called the *shell*. Commands that were not transported from UNIX System V reside in **/usr/local/bin**; this directory is recommended for locally implemented programs. Some administrative commands reside in **/etc** and various other places. The **/etc** directory is searched automatically if you are logged in as root; otherwise use the full path name given under **SYNOPSIS** or change the **PATH** environment variable to include the command's directory.

Section 2, System Calls, describes the entries into the CTIX kernel, including the C language interfaces.

Section 3, Subroutines and Libraries, describes the available library functions or subroutines. Their binary versions reside in various system libraries in the directories **/lib** and **/usr/lib**. See *intro*(3) for descriptions of these libraries and the files in which they are stored.

Section 4, File Formats, documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in *a.out*(4). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language **struct** declarations corresponding to these formats can be found in the directories **/usr/include** and **/usr/include/sys**.

Section 5, Miscellaneous Facilities, contains descriptions of character sets, macro packages, and other such information.

Section 6, Games, describes the games and educational programs that reside in the directory **/usr/games**.

Section 7, Special Files, discusses the characteristics of files that actually refer to input/output devices.

Entry organization. All entries are based on a common format, in which some parts are optional:

NAME The **NAME** part gives the name(s) of the entry and briefly states its purpose.

SYNOPSIS The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (Commands and Application Programs):

Bold **Boldface** strings are literals, and are to be typed just as they appear.

Regular *Regular face* strings usually represent substitutable argument prototypes and program names found elsewhere in the manual.

[ ] Square brackets around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file," it always refers to a *file* name.

... Ellipses are used to show that the previous argument prototype can be repeated.

– + = A final convention is used by the commands themselves. An argument beginning with a minus (–), plus (+), or equal sign (=) is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with –, +, or =.

DESCRIPTION The **DESCRIPTION** part discusses the subject at hand.

EXAMPLE(S) The **EXAMPLE(S)** part gives example(s) of usage, where appropriate.

FILES The **FILES** part gives the file names that are built into the program.

SEEALSO The **SEE ALSO** part gives pointers to related information.

DIAGNOSTICS The **DIAGNOSTICS** part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

NOTES The **NOTES** part gives information that might be helpful under the particular circumstance described.

WARNINGS The **WARNINGS** part points out potential pitfalls.

BUGS The **BUGS** part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents is provided at the front of each of the four volumes, along with a complete permuted index derived from the tables. On each *index* line, the title of the

entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call.

PERMUTED INDEX

This index includes entries for all pages of Volumes 1 through 4. The entries themselves are based on the one-line descriptions or titles found in the NAME portion of each manual page; the significant words (keywords) of these descriptions are listed alphabetically down the center of the index.

The index is actually a keyword-in-context (KWIC) index that has three columns. To use the index, read the center column to look up specific commands by name or by subject topics. Note that the entry may begin in the left column or wrap around and continue into the left column. A period (.) marks the end of the entry, and a slash (/) indicates where the entry has been continued or truncated. The right column gives the manual page where the command or subject is described.

NAME

m4 - macro processor

SYNOPSIS

m4 [options] [files]

DESCRIPTION

The *m4* command is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is -, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

-e Operate interactively. Interrupts are ignored and the output is unbuffered.

-s Enable line sync output for the C preprocessor (#line ...)

-B*int* Change the size of the push-back and argument collection buffers from the default of 4096.

-H*int* Change the size of the symbol table hash array from the default of 199. The size should be prime.

-S*int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

-T*int* Change the size of the token buffer from the default of 512 bytes.

To be effective, the options listed above must appear before any file names and before any **-D** or **-U** flags.

-D*name*[=*val*]
Defines *name* to *val* or to null in *val*'s absence.

-U*name* undefines *name*.

Macro calls have the following form:

name(arg1,arg2, ..., argn)

The left parenthesis (( ) must immediately follow the name of the macro. If the name of a defined macro is not followed by a left parenthesis, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore (_), where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotation marks are used to quote strings. The value of a quoted string is the string stripped of the quotation marks.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

The *m4* command makes available the following built-in macros. They can be redefined, but in doing so, the original meaning is lost. Unless otherwise stated, the macro values are null.

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; $# is replaced by the number of arguments; $* is replaced by a list of all the arguments separated by commas; $@ is like $*, but each argument is quoted (with the current quotation marks).

undefine — Removes the definition of the macro named in its argument.

defn — Returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

pushdef — Like *define*, but saves any previous definition.

popdef — Removes current definition of its argument(s), exposing the previous one, if any.

ifdef — If the first argument is defined, the value is the second argument; otherwise, it is the third. If there is no third argument, the value is null. The word *unix* is predefined on CTIX system versions of *m4*.

shift — Returns all but its first argument; the other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that is subsequently performed.

changequote — Changes quotation symbols to the first and second arguments. The symbols can be up to five characters long. *Changequote* without arguments restores the original values (that is, ` ').

changecom — Changes left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is

effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes newline. With two arguments, both markers are affected. Comment markers may be up to five characters long.

divert — *m4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert — Causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text can be undiverted into another diversion. Undiverting discards the diverted text.

divnum — Returns the value of the current output stream.

dnl — Reads and discards characters up to and including the next new-line.

ifelse — Takes three or more arguments. If the first argument is the same string as the second, the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7; otherwise, the value is either the fourth string, or, if it is not present, null.

incr — Returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

decr — Returns the value of its argument decremented by 1.

eval — Evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include the symbols +, -, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; and parentheses. Octal and hexadecimal numbers can be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument can be used to specify the minimum number of digits in the result.

len — Returns the number of characters in its argument.

index — Returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.

substr	Returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
translit	Transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	Returns the contents of the file named in the argument.
sinclude	Identical to *include*, except that it does not report if the file is inaccessible.
syscmd	Executes the CTIX system command given in the first argument. No value is returned.
sysval	The return code from the last call to *syscmd*.
maketemp	Fills in a string of XXXXX in its argument with the current process ID.
m4exit	Causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.
m4wrap	Pushes back argument 1 at final EOF; for example, **m4wrap(`cleanup()')**
errprint	Prints its argument on the diagnostic output file.
dumpdef	Prints current names and definitions for the named items, or for all if no arguments are given.
traceon	With no arguments, enables tracing for all macros (including built-ins); otherwise, enables tracing for named macros.
traceoff	Disables trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

SEE ALSO

cc(1), cpp(1).
Programmer's Guide: CTIX Supplement.

NAME

machid: mc68k, miti, mini, mega, unixpc, i386, i286, pdp11, u3b, u3b2, u3b5, u3b15, u370, vax - get processor type truth value

SYNOPSIS

mc68k

miti

mini

mega

unixpc

i386

i286

pdp11

u3b

u3b2

u3b5

u3b15

u370

vax

DESCRIPTION

The following commands return a true value (exit code of 0) if you are on a processor that the command name indicates.

mc68k True if you are on a 68000-, 68010-, or 68020-based computer.

miti True if you are on an S/Series computer.

mini True if you are on a MiniFrame computer.

mega True if you are on an S/1280 computer.

unixpc True if you are on a Unix PC computer.

i386 True if you are on an Intel 80386-based computer.

i286 True if you are on an Intel 80286-based computer.

pdp11 True if you are on a PDP-11/45 or PDP-11/70.

u3b True if you are on a 3B20 computer.

u3b2 True if you are on a 3B2 computer.

u3b5 True if you are on a 3B5 computer.

u3b15 True if you are on a 3B15 computer.

u370 True if you are on an IBM 370 computer.

vax True if you are on a VAX-11/750 or VAX-11/780.

The commands that do not apply will return a false (non-zero) value. These commands are often used within makefiles [see *make*(1)] and shell procedures [see *sh*(1)] to increase portability.

SEE ALSO

make(1), sh(1), test(1), true(1).

NAME

mail, rmail - send mail to users or read mail

SYNOPSIS

Sending mail:

mail [**-wt**] persons

rmail [**-wt**] persons

Reading mail:

mail [**-ehpqr**] [**-f** file] [**-F** persons]

DESCRIPTION

Sending mail:

The command-line arguments that follow affect SENDING mail:

-w causes a letter to be sent to a remote user without waiting for the completion of the remote transfer program.

-t causes a **To:** line to be added to the letter, showing the intended recipients.

A *person* is usually a user name recognized by *login*(1). When *persons* are named, *mail* assumes a message is being sent (except in the case of the -F option). It reads from the standard input up to an end-of-file (control-d), or until it reads a line consisting of just a period. When either of those signals is received, *mail* adds the *letter* to the *mailfile* for each *person*. A *letter* is a *message* preceded by a *postmark*. The message is preceded by the sender's name and a *postmark*. A *postmark* consists of one or more 'From' lines followed by a blank line.

If a letter is found to be undeliverable, it is returned to the sender with diagnostics that indicate the location and nature of the failure. If *mail* is interrupted during input, the file **dead.letter** is saved to allow editing and resending. **dead.letter** is recreated every time it is needed, erasing any previous contents.

rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

If the local system has been configured for UUCP, mail can be sent to a recipient on a remote system. Prefix *person* by the system name and exclamation point. A series of system names separated by exclamation points can be used to direct a letter through an extended network.

Reading Mail:

The command-line arguments that follow affect reading mail:

-e Causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.

-h Causes a window of headers to be displayed rather than the latest message. The display is followed by the **?** prompt.

-p Causes all messages to be printed without prompting for disposition.

-q Causes *mail* to terminate after interrupts. Normally an interrupt causes only the termination of the message being printed.

-r Causes messages to be printed in first-in, first-out order.

-f*file* Causes *mail* to use *file* (for example, **mbox**) instead of the default *mailfile*.

-F*persons*
Entered into an empty *mailbox*, causes all incoming mail to be forwarded to *persons*.

Unless otherwise influenced by command-line arguments, *mail* prints a user's mail messages in last-in, first-out order. For each message, the user is prompted with a **?**, and a line is read from the standard input. The following commands are available to determine the disposition of the message:

<newline>, **+**, or **n**	Go on to next message.
d, or **dp**	Delete message and go on to next message.
d #	Delete message number #. Do not go on to next message.
dq	Delete message and quit *mail*.
h	Display a window of headers around current message.
h #	Display header of message number #.
h a	Display headers of ALL messages in the user's *mailfile*.
h d	Display headers of messages scheduled for deletion.
p	Print current message again.
-	Print previous message.
a	Print message that arrived during the *mail* session.

#	Print message number #.
r [*users*]	Reply to the sender, and other *user(s)*, then delete the message.
s [*files*]	Save message in the named *files* (**mbox** is default).
y	Same as save.
u [#]	Undelete message number # (default is last read).
w [*files*]	Save message, without its top-most header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons*.
q, or **ctl-d**	Put undeleted mail back in the *mailfile* and quit *mail*.
x	Put all mail back in the *mailfile* unchanged and exit from *mail*.
!*command*	Escape to the shell to do *command*.
?	Print a command summary.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

The *mailfile* can be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file can also contain the following first line which forwards to *person* all mail sent to the owner of the *mailfile*:

Forward to *person*

A *Forwarded by...* message is added to the header. This is especially useful in a multi-machine environment to forward all of a person's mail to a single machine, and to keep the recipient informed if the mail has been forwarded. Installation and removal of forwarding is done with the **-F** option.

To forward all of one's mail to systema!user, enter:

```
mail -Fsystema!user
```

To forward to more than one user, enter:

```
mail -F"user1,systema!user2,systema!systemb!user3"
```

Note that when more than one user is specified, the whole list should be enclosed in double quotation marks so that it can all be interpreted as the

operand of the **-F** option. The list can be up to 1024 bytes; either commas or white space can be used to separate users.

To remove forwarding enter:

```
mail -F ""
```

The pair of double quotation marks is mandatory to set a NULL argument for the -F option.

In order for forwarding to work properly the *mailfile* should have **mail** set as group ID, and the group permission should be read-write.

Note that *mail* can not be used as a user agent for *sendmail*.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*user*	incoming mail for *user*; that is, the *mailfile*
$HOME/mbox	saved mail
$MAIL	variable containing path name of *mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), mailx(1), write(1).
UNIX System V Release 3.2 User's Guide.
S/Series CTIX Administrator's Guide.
CTIX Administration Tools Manual.

WARNING

The "Forward to person" feature can result in a loop, if *sys1!userb* forwards to *sys2!userb* and *sys2!userb* forwards to *sys1!userb*. The symptom is the display of the following message:

unbounded...saved mail in dead.letter.

BUGS

Conditions sometimes result in a failure to remove a lock file.

After an interrupt, the next message might not be printed; enter **p** to force printing.

NAME

mailx - interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Many of the remote features of *mailx* work only if UUCP is configured on your system. *mailx* can also act as a user agent for *sendmail* (see the **sendmail** variable below).

Incoming mail is stored in a standard file for each user, called the **mailbox** for that user. When *mailx* is called to read messages, the **mailbox** is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the **mbox**, and is normally located in the user's HOME directory (see the description of MBOX under ENVIRONMENT VARIABLES for a Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until manually removed.

The user can access a secondary file by using the **-f** option of the *mailx* command. Messages in the secondary file can then be read or otherwise processed by using the same commands used in the primary **mailbox**. This gives rise within these pages to the notion of a current **mailbox**.

On the command line, *options* start with a dash (-) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* attempts to read messages from the **mailbox**. Command line options are as follows:

-e Test for presence of mail. *mailx* prints nothing and exits with a successful return code if there is mail to read.

-f [*filename*] Read messages from *filename* instead of from **mailbox**. If no *filename* is specified, the **mbox** is used.

-F Record the message in a file named after the first recipient. Overrides the **record** variable, if set; see the desctiption of **record** under ENVIRONMENT VARIABLES.

-h *number*	The number of network hops made so far. This is provided for network software to avoid infinite delivery loops; see the description of **addsopt** under ENVIRONMENT VARIABLES.
-H	Print header summary only.
-i	Ignore interrupts; see the description of **ignore** under (ENVIRONMENT VARIABLES).
-n	Do not initialize from the system default **mailx.rc** file.
-N	Do not print initial header summary.
-r *address*	Pass *address* to network delivery software. All tilde commands are disabled; see the description of **addsopt** under ENVIRONMENT VARIABLES.
-s *subject*	Set the Subject header field to *subject*.
-u *user*	Read *user*'s **mailbox**. This is effective only if *user*'s **mailbox** is not read-protected.
-U	Convert *uucp* style addresses to Internet standards. Overrides the **conv** environment variable; see the description of **addsopt** under ENVIRONMENT VARIABLES.

When reading mail, *mailx* is in *command mode*. A header summary of the first several messages displays, followed by a prompt indicating that *mailx* can accept regular commands; see COMMANDS below. When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject appears. (A subject longer than 1024 characters causes *mailx* to dump core.) As the message is typed, *mailx* reads the message and stores it in a temporary file. Commands can be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the **set** and **unset** commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If mail is found to be undeliverable, an attempt is made to return it to the sender's *mailbox*. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface

with any program that reads the standard input, such as *lp*(1) for recording outgoing mail on paper. Alias groups are set by the **alias** command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, **print** is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many commands take an optional list of messages (*msglist*) to operate on. The default for *msglist* is the current message. A *msglist* is a list of message identifiers separated by spaces, which may include:

n Message number **n**.

. The current message.

^ The first undeleted message.

$ The last message.

***** All messages.

n-m An inclusive range of message numbers.

user All messages from **user**.

/string All messages with **string** in the subject line (case ignored).

:c All messages of type *c*, where *c* is one of:

d deleted messages
n new messages
o old messages
r read messages
u unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions [see *sh*(1)]. Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* tries to execute commands from the optional system-wide file (**/usr/lib/mailx/ mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. With the exceptions noted below, regular commands are legal inside start-up files. The most common use of a start-up file is to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, Copy, edit, **f**ollowup, **F**ollowup, **h**old, **m**ail, **pr**eserve, **r**eply, **R**eply, **sh**ell, and visual. An error in the start-up file causes the remaining lines in the file to be ignored. The **.mailrc** file is optional and must be constructed locally.

COMMANDS

The following is a complete list of *mailx* commands:

!*shell-command*

Escape to the shell. See "SHELL" (ENVIRONMENT VARIABLES).

comment

Null command (comment). This may be useful in **.mailrc** files.

=

Print the current message number.

?

Prints a summary of commands.

alias *alias name ...*

group *alias name ...*

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the **.mailrc** file.

alternates *name ...*

Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, **alt**ernates prints the current list of alternate names. See also "allnet" (ENVIRONMENT VARIABLES).

cd [*directory*]

chdir [*directory*]

Change directory. If *directory* is not specified, $HOME is used.

copy [*filename*]

copy [*msglist*] *filename*

Copy messages to the file without marking the messages as saved. Otherwise, equivalent to the save command.

Copy [*msglist*]

Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the Save command.

delete [*msglist*]

Delete messages from the **mailbox**. If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES).

discard [*header-field* ...]

ignore [*header-field* ...]

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The **P**rint and **T**ype commands override this command.

dp [*msglist*]

dt [*msglist*]

Delete the specified messages from the **mailbox** and print the next message after the last one deleted. Roughly equivalent to a **d**elete command followed by a **p**rint command.

echo *string* ...

Echo the given strings [like *echo*(1)].

edit [*msglist*]

Edit the given messages. The messages are placed in a temporary file and the "EDITOR" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES). Default editor is *ed*(1).

exit

xit

Exit from *mailx*, without changing the **mailbox**. No messages are saved in the **m**box (see also **q**uit).

file [*filename*]

folder [*filename*]

Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

%	the current **mailbox**.
%**user**	the **mailbox** for **user**.
#	the previous file.
&	the current **mbox**.

Default file is the current **mailbox**.

folders

Print the names of the files in the directory set by the "folder" variable (see ENVIRONMENT VARIABLES).

followup [*message*]

Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the "record" variable, if set. See also the **F**ollowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

Followup [*msglist*]

Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the **f**ollowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

from [*msglist*]

Prints the header summary for the specified messages.

group *alias name ...*

alias *alias name ...*

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the **.mailrc** file.

headers [*message*]

Prints the page of headers which includes the message specified. The "screen" variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the **z** command.

help

Prints a summary of commands.

hold [*msglist*]
preserve [*msglist*]
Holds the specified messages in the **mailbox**.

if *s* | *r*
mail-commands
else
mail-commands
endif
Conditional execution, where *s* will execute following *mail-commands*, up to an **else** or **en**dif, if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the **.mailrc** file.

ignore *header-field ...*
discard *header-field ...*
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The **P**rint and **T**ype commands override this command.

list
Prints all commands available. No explanation is given.

mail *name ...*
Mail a message to the specified users.

Mail *name*
Mail a message to the specified user and record a copy of it in a file named after that user.

mbox [*msglist*]
Arrange for the given messages to end up in the standard **mbox** save file when *mailx* terminates normally. See "MBOX" (ENVIRONMENT VARIABLES) for a description of this file. See also the **e**xit and **q**uit commands.

next [*message*]
Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user,

since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]
| [*msglist*] [*shell-command*]
Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

preserve [*msglist*]
hold [*msglist*]
Preserve the specified messages in the **mailbox**.

Print [*msglist*]
Type [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **i**gnore command.

print [*msglist*]
type [*msglist*]
Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

quit
Exit from *mailx*, storing messages that were read in **m**box and unread messages in the **mailbox**. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]
Respond [*msglist*]
Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

reply [*message*]

respond [*message*]

Reply to the specified message, including all other recipients of the message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

Save [*msglist*]

Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the Copy, followup, and Followup commands and "outfolder" (ENVIRONMENT VARIABLES).

save [*filename*]

save [*msglist*] *filename*

Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the **mailbox** when *mailx* terminates unless "keepsave" is set (see also ENVIRONMENT VARIABLES and the exit and quit commands).

set

set *name*

set *name=string*

set *name=number*

Define a variable called *name*. The variable may be given a null, string, or numeric value. Set by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the *mailx* variables.

shell

Invoke an interactive shell [see also "SHELL" (ENVIRONMENT VARIABLES)].

size [*msglist*]

Print the size in characters of the specified messages.

source *filename*

Read commands from the given file and return to command mode.

top [*msglist*]
Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

touch [*msglist*]
Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the **mbox**, or the file specified in the MBOX environment variable, upon normal termination. See **exit** and **quit**.

Type [*msglist*]
Print [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ignore** command.

type [*msglist*]
print [*msglist*]
Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

undelete [*msglist*]
Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

unset *name ...*
Causes the specified variables to be erased. If the variable was imported from the execution environment (that is, a shell variable) then it cannot be erased.

version
Prints the current version and release date.

visual [*msglist*]
Edit the given messages with a screen editor. The messages are placed in a temporary file and the "VISUAL" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

write [*msglist*] *filename*
Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the save command.

xit

exit

Exit from *mailx*, without changing the **mailbox**. No messages are saved in the **mbox** (see also quit).

z[+ | -]

Scroll the header display forward or backward one screen-full. The number of headers displayed is set by the "screen" variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands can be entered only from *input mode*, by beginning a line with the tilde escape character (~). See "escape" (ENVIRONMENT VARIABLES) for changing this special character.

~! *shell-command*

Escape to the shell.

~.

Simulate end of file (terminate message input).

~: *mail-command*

~_ *mail-command*

Perform the command-level request. Valid only when sending a message while reading mail.

~?

Print a summary of tilde escapes.

~**A**

Insert the autograph string "Sign" into the message (see ENVIRONMENT VARIABLES).

~**a**

Insert the autograph string "sign" into the message (see ENVIRONMENT VARIABLES).

~**b** *name* ...

Add the *name*s to the blind carbon copy (Bcc) list.

~**c** *name* ...

Add the *name*s to the carbon copy (Cc) list.

~**d**

Read in the **dead.letter** file. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~e

Invoke the editor on the partial message. See also "EDITOR" (ENVIRONMENT VARIABLES).

~f [*msglist*]

Forward the specified messages. The messages are inserted into the message without alteration.

~h

Prompt for Subject line and To, Cc, and Bcc lists. If the field displays with an initial value, it may be edited as if you had just typed it.

~i *string*

Insert the value of the named variable into the text of the message. For example, ~A is equivalent to '~i Sign.' Environment variables set and exported in the shell are also accessible by ~i.

~m [*msglist*]

Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~p

Print the message being entered.

~q

Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in **dead.letter**. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~r *filename*

~~< *filename*

~~< !*shell-command*

Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

~s *string* ...

Set the subject line to *string*.

~t *name* ...

Add the given *name*s to the To list.

~v

Invoke a preferred screen editor on the partial message. See also "VISUAL" (ENVIRONMENT VARIABLES).

~w *filename*

Write the partial message onto the given file, without the header.

~x

Exit as with ~q except the message is not saved in **dead.letter**.

~| *shell-command*

Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within *mailx*.

HOME=*directory*

The user's base of operations.

MAILRC=*filename*

The name of the start-up file. Default is $HOME/.mailrc.

The following variables are internal *mailx* variables. They can be imported from the execution environment or set by using the **set** command at any time. The **unset** command can be used to erase variables.

addsopt

Enabled by default. If */bin/mail* is not being used as the deliverer, **noaddsopt** should be specified. (See WARNINGS below)

allnet

All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet**. See also the **alt**ernates command and the "metoo" variable.

append

Upon termination, append messages to the end of the **mbox** file instead of prepending them. Default is **noappend.**

askcc

Prompt for the Cc list after message is entered. Default is **noaskcc**.

asksub

Prompt for subject if it is not specified on the command line with the -s option. Enabled by default.

autoprint

Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint**.

bang

Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(1). Default is **nobang**.

cmd=*shell-command*

Set the default command for the **pipe** command. No default value.

conv=*conversion*

Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC 822 standard for electronic mail addressing. Conversion is disabled by default. See also "sendmail" and the **-U** command line option.

crt=*number*

Pipe messages having more than *number* lines through the command specified by the value of the "PAGER" variable [*pg*(1) by default]. Disabled by default.

DEAD=*filename*

The name of the file in which to save partial letters in case of untimely interrupt. Default is $HOME/dead.letter.

debug

Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

dot

Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot**.

EDITOR=*shell-command*

The command to run when the **edit** or **~e** command is used. Default is *ed*(1).

escape=*c*

Substitute *c* for the ~ escape character. Takes effect with next message sent.

folder=*directory*

The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a *mailx* command line, "folder" must be an exported *sh* environment variable. There is no default for the "folder" variable. See also "outfolder" below.

header

Enable printing of the header summary when entering *mailx*. Enabled by default.

hold

Preserve all messages that are read in the **mailbox** instead of putting them in the standard **mbox** save file. Default is **nohold**.

ignore

Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof

Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the ~. command. Default is **noignoreeof**. See also "dot" above.

keep

When the **mailbox** is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave

Keep messages that have been saved in other files in the **mailbox** instead of deleting them. Default is **nokeepsave**.

MBOX=*filename*

The name of the file to save messages which have been read. The xit command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

metoo

If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

LISTER=*shell-command*

The command (and options) to use when listing the contents of the "folder" directory. The default is *ls*(1).

onehop

When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (that is, one hop away).

outfolder

Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the path name is absolute. Default is **nooutfolder**. See "folder" above and the Save, Copy, followup, and Followup commands.

page

Used with the **pipe** command to insert a form feed after each message sent through the pipe. Default is **nopage**.

PAGER=*shell-command*

The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg*(1).

prompt=*string*

Set the *command mode* prompt to *string*. Default is "? ".

quiet

Refrain from printing the opening message and version when entering *mailx*. Default is **noquiet**.

record=*filename*

Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save

Enable saving of messages in **dead.letter** on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen=*number*

Sets the number of lines in a screen-full of headers for the headers command.

sendmail=*shell-command*

Alternate command for delivering messages. Default is */bin/rmail*. See *mail*(1).

On systems running *sendmail*, the system administrator should add the command

```
set sendmail = /usr/lib/sendmail
```

in the system-wide *mailx* startup file, **/usr/lib/mailx/mailx.rc**.

sendwait

Wait for background mailer to finish before returning. Default is **nosendwait**.

SHELL=*shell-command*

The name of a preferred command interpreter. Default is *sh*(1).

showto

When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*string*

The variable inserted into the text of a message when the ~**a** (autograph) command is given. No default [see also ~**i** [TILDE ESCAPES)].

Sign=*string*

The variable inserted into the text of a message when the ~**A** command is given. No default [see also ~**i** [TILDE ESCAPES)].

toplines=*number*

The number of lines of header to print with the **top** command. Default is 5.

VISUAL=*shell-command*

The name of a preferred screen editor. Default is *vi*(1).

FILES

$HOME/.mailrc	personal start-up file
$HOME/mbox	secondary storage file
/usr/mail/*	post office directory
/usr/lib/mailx/mailx.help*	help message files
/usr/lib/mailx/mailx.rc	optional global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO

ls(1), mail(1), pg(1), sendmail(1M).

WARNINGS

The **-h**, **-r** and **-U** options can be used only if *mailx* is built with a delivery program other than */bin/mail*. The tilde character (~) does not work as the escape character over the network; substitute another character for escape (see ESCAPE under ENVIRONMENT VARIABLES).

BUGS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

The full internet addressing is not fully supported by *mailx*. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a "." are treated as the end of the message by *mail*(1) (the standard mail delivery program).

NAME

make - maintain, update, and regenerate groups of programs

SYNOPSIS

make [**-f** makefile] [**-p**] [**-i**] [**-k**] [**-s**] [**-r**] [**-n**] [**-b**] [**-e**] [**-u**] [**-t**] [**-q**] [names]

DESCRIPTION

The *make* command allows the programmer to maintain, update, and regenerate groups of computer programs. The following is a brief description of all options and some special names:

-f *makefile* Description filename. *makefile* is assumed to be the name of a description file.

-p Print out the complete set of macro definitions and target descriptions.

-i Ignore error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.

-k Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.

-s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.

-r Do not use the built-in rules.

-n No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.

-b Compatibility mode for old makefiles.

-e Environment variables override assignments within makefiles.

-t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

-q Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

.DEFAULT If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.

.PRECIOUS
Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT Same effect as the **-s** option.

.IGNORE Same effect as the **-i** option.

make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **-f** option is present, **makefile**, **Makefile**, and the Source Code Control System (SCCS) files **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is **-**, the standard input is taken. More than one **-f** *makefile* argument pair may appear.

make updates a target only if its dependents are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date.

makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a :, then a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that begin with a tab are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands can be continued across lines with the <backslash><new-line> sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (#) and new-line surround comments.

The following *makefile* says that **pgm** depends on two files, **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o: incl.h a.c
        cc -c a.c
b.o: incl.h b.c
        cc -c b.c
```

Command lines are executed one at a time, each by its own shell. The SHELL environment variable can be used to specify which shell *make* should use to execute commands. The default is */bin/sh*. The first one or two characters in a command can be the following: -, @, -@, or @-. If @ is present, printing of the command is suppressed. If - is present, *make* ignores an error. A line is printed when it is executed unless the **-s** option is present, or the entry **.SILENT:** is in *makefile*, or unless the initial character sequence contains a @. The **-n** option specifies printing without execution; however, if the command line has the string $(MAKE) in it, the line is always executed (see discussion of the MAKEFLAGS macro under *Environment*). The **-t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the **-i** option is present, or the entry **.IGNORE:** appears in *makefile*, or the initial character sequence of the command contains -, the error is ignored. If the **-k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **-b** option allows old makefiles (those written for the old version of *make*) to run without errors.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name **.PRECIOUS**.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **-e** option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built-in rules.

The MAKEFLAGS environment variable is processed by *make* as containing any legal input option (except **-f** and **-p**) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, MAKEFLAGS always contains the current input options. This proves very useful for "super makes." In fact, as noted above, when the **-n** option is

used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make -n** recursively on a whole software system to see what would have been executed. This is because the **-n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Include Files

If the string *include* appears as the first seven letters of a line in a *makefile*, and is followed by a blank or a tab, the rest of the line is assumed to be a filename and will be read by the current invocation, after substituting for any macros.

Macros

Entries of the form *string1 = string2* are macro definitions. *String2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of $(*string1* [:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under Libraries.

Internal Macros

There are five internally maintained macros that are useful for writing rules for building targets.

$*** The macro **$* stands for the filename part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@** The **$@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$<** The **$< macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module that is out-of-date with respect to the target (that is, the "manufactured" dependent filename). Thus, in the **.c.o** rule, the **$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
        cc -c -O $*.c
```

or:

```
.c.o:
        cc -c -O $<
```

$?** The **$? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules that must be rebuilt.

$% The **$%** macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **$@** evaluates to **lib** and **$%** evaluates to the library member, **file.o**.

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros, the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **$?**.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules that allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c~ .f .f~ .sh .sh~
.c.o .c.a .c~.o .c~.c .c~.a
.f.o .f.a .f~.o .f~.f .f~.a
.h~.h .s.o .s~.o .s~.s .s~.a .sh~.sh
.l.o .l.c .l~.o .l~.l .l~.c
.y.o .y.c .y~.o .y~.y .y~.c
```

The internal rules for *make* are contained in the source file **rules.c** for the *make* program. These rules can be locally modified. To print the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used:

```
make -fp - 2>/dev/null </dev/null
```

A tilde in the above rules refers to an SCCS file [see *sccsfile* (4)]. Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (that is, **.c:**) is the definition of how to build *x* from *x***.c**. In effect, the other suffix is null. This is useful for building targets from only one source file (for example, shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f .f~

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

```
pgm: a.o b.o
        cc a.o b.o -o pgm a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user can add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus, **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library that contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form **.XX.***a* where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, you cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date
.c.a:   $(CC) -c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        $(AR) $(ARFLAGS) lib $?
        rm $? @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object filenames (inside **lib**) whose C source files are out-of-date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

FILES

[Mm]akefile and s.[Mm]akefile
/bin/sh

SEE ALSO

cc(1), cd(1), lex(1), sh(1), yacc(1), printf(3S), sccsfile(4).
UNIX System V Release 3.2 Programmer's Guide.

NOTES

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty.

BUGS

Filenames with the characters = : @ will not work. Commands that are directly executed by the shell, notably *cd*(1), are ineffective across newlines in *make*. The syntax **(lib(file1.o file2.o file3.o)** is illegal. You cannot build **lib(file.o)** from **file.o**. The macro **$(a:.o=.c~)** does not work. Named pipes are not handled well.

NAME

makekey - generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

The *makekey* command improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (that is, to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and uppercase and lowercase letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

makekey is intended for programs that perform encryption. Usually, its input and output will be pipes.

SEE ALSO

ed(1), crypt(1), vi(1), passwd(4).

CAVEAT

The *makekey* command can produce different results depending upon whether the input is typed at the terminal or redirected from a file.

WARNING

The standard CTIX distribution is the international version, which does not support encryption.

NAME

masterupd - update the master file

SYNOPSIS

masterupd [flags] [devicename]

masterupd [**-t**] [**-T**] [flags] [parameter
[definename definevalue minvalue masvalue]]

DESCRIPTION

The *masterupd* command is used to modify the **/etc/master** file. Different options to *masterupd* add/delete devices in the file. Options are also available to add/delete tunable parameters in the master file. Although the file can be directly edited, it is often more convenient to use the features of *masterupd*.

The options to *masterupd* follow:

-a Add a device (or, if **-t** or **-T** is used, a tunable parameter) to the master file. The *devicename* must be specified. Also, the **-f** option must be given to specify the device's functions.

-d Delete a device (or, if **-t** or **-T** is used, a tunable parameter) from the master file. Only the *devicename* need be given. If the device is not found in the master file, the command silently terminates with no error message.

-q Query the master file for a particular device. The *devicename* must be given. Also, one of the device type options (see below) must be given to specify the type of device to look for. If a block or character device is specified, the major number is written to standard output. If a line discipline is specified, the line discipline index is output. If a software module is specified, the software ID is output. If the device cannot be found in the master file, *masterupd* returns a question mark.

-v Verify the consistency of the master file.

-M Specify an alternate master file.

Any or all of the above options can be used together. If **-a** is used with **-d**, the device is deleted from the master file first, then added again. If the **-q** option is used with **-a** and/or **-d**, it reports the major number of the device after the addition/deletion.

The **-a** option and the **-q** option require a device type to be given. The following options specify the device type:

-b block device

-c character device

-l line discipline

-m stream module

-s software module

-r required device

-F file system

Other options that can be used with **-a** follow:

-f followed by a string of function letters, specifies functions for the device. See *master*(4) for a description of the letters.

-p followed by a string, specifies a prefix for the device. The default prefix is the device name.

-V followed by a decimal integer, specifies an interrupt vector for the device.

-L followed by a decimal integer, specifies an interrupt priority level for the device.

-i followed by a decimal integer, specifies an init parameter for the device. Up to three **-i** options can specify up to three init parameters, which are appended to the end of the line in the master file.

For file systems, exactly two init parameters must be given. The first becomes the *flags* field; the second becomes the *notify* field of the **fsinfo** structure.

The following options (used with the **-a** or **-d** option, described earlier) specify that you want to add/delete a tunable parameter in the master file. Depending on which you use (**-t** or **-T**), you can use *parameter* or *definename* to specify the tunable parameter to add/delete.

-t Specifies that you want to add/delete a tunable parameter in the master file. When adding a parameter with **-t**, all three fields (*parameter*, *definename*, *definevalue*) must be present. The *minvalue* and *maxvalue* arguments are optional; if not specified, the parameter is added to the **master** file without min or max values.

When deleting a parameter, only *parameter* should be given. If a parameter with that parameter name already exists in the **master** file, an error is reported.

If *parameter* is a dash (-), and there is an existing parameter with the same *definename*, the parameter name is taken to be the parameter name of the existing parameter. If there is no existing parameter, the parameter name is taken to be the same as the *definename*.

-T Like the **-t** option, except that **-T** uses *definename* instead of *parameter* to find the tunable parameter to add/delete.

When adding a tunable parameter, **-T** acts like **-t**, except that an error is reported if an existing parameter in the **master** file has the same define name (rather than the same parameter name).

When deleting, the tunable parameter whose define name matches *definename* is deleted. The single argument is taken to be the define name rather than the parameter name.

EXAMPLES

The following command installs a new character device driver named "xyz" (with the prefix **xyz**) and functions open, close, read, write, ioctl, init, and release:

```
masterupd -a -c -f ocrwi xyz
```

As a more realistic example, the following script does the same as the command shown above, but it first deletes the driver if it already exists in the master file and then it uses the query feature to make **/dev** files for the newly assigned character major number (note that the single quotation marks must be backquotes):

```
MAJ=`masterupd -daq -c -f ocrwi xyz`
if [ "$MAJ" = "?" ]
then
        echo "Error"
else
        mknod /dev/xyz0 c $MAJ 0
        mknod /dev/xyz1 c $MAJ 1
        ...
fi
```

SEE ALSO

config(1M), master(4).

NAME

mcs - manipulate the object file comment section

SYNOPSIS

mcs [options] object-file ...

DESCRIPTION

The *mcs* command manipulates the comment section, normally the ".comment" section, in an object file. It is used to add to, delete, print, and compress the contents of the comment section in a CTIX System object file. *mcs* must be given one or more of the options described below. It takes each of the options given and applies them in order to the *object-files*.

If the object file is an archive, the file is treated as a set of individual object files. For example, if the **-a** option is specified, the string is appended to the comment section of each archive element.

The following options are available.

-a *string*
: Append *string* to the comment section of the *object-files*. If *string* contains embedded blanks, it must be enclosed in quotation marks.

-c
: Compress the contents of the comment section. All duplicate entries are removed. The ordering of the remaining entries is not disturbed.

-d
: Delete the contents of the comment section from the object file. The object file comment section header is removed also.

-n *name*
: Specify the name of the section to access. By default, *mcs* deals with the section named *.comment*. This option can be used to specify another section.

-p
: Print the contents of the comment section on the standard output. If more than one name is specified, each entry printed is tagged by the name of the file from which it was extracted, using the format "filename:string."

EXAMPLES

To print a file's *comment* section:

```
mcs -p file
```

To append *string* to *file*'s *comment* section:

```
mcs -a string file
```

FILES

TMPDIR/mcs*	temporary files
TMPDIR/*	temporary files

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

SEE ALSO

cpp(1), a.out(4).

NOTES

mcs cannot insert or delete comment sections in executable objects with magic number 0413. (By default, *ld* creates executable objects with magic number 0413 [see *a.out*(4) and *ld*(1)].) However, the **-d** option to *mcs* can make the comment section of a 0413 file zero-length: this allows use of *mcs* with the **-a** and **-c** options on such files. All libraries provided with CTIX have comment sections in each library member as well as in the C start up routines: thus, there would normally be a comment section in every **a.out** file.

NAME

mesg - permit or deny messages

SYNOPSIS

mesg [**-n**] [**-y**]

DESCRIPTION

mesg with argument **n** forbids messages via *write*(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).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdbsym - load symbols in kernel debugger

SYNOPSIS

/etc/lddrv/mkdbsym [**-f** input file] [**-o** output file]

DESCRIPTION

mkdbsym takes an input file (default **/unix**) and writes the symbols to the output file (default **/dev/dbsym**, in which case the running kernel is changed). The drivers **debugger** and **dbsym** must be in the kernel [these drivers may be loaded with *lddrv*(1M), if they are not already loaded].

FILES

/unix
/dev/dbsym

SEE ALSO

lddrv(1M), dbsym(7).

NAME

mkdir, mkdirs - make directories

SYNOPSIS

mkdir [**-m** mode] [**-p**] dirname ...

mkdirs [**-e**] dirname ...

DESCRIPTION

mkdir creates the named directories in mode 777 [possibly altered by *umask*(1)].

Standard entries in a directory (for example, the files ., for the directory itself, and .., for its parent) are made automatically. *mkdir* cannot create these entries by name. Creation of a directory requires write permission in the parent directory.

The owner ID and group ID of the new directories are set to the process's real user ID and group ID, respectively.

Two options apply to *mkdir*:

-m This option allows users to specify the mode to be used for new directories. Choices for modes can be found in *chmod*(1).

-p With this option, *mkdir* creates *dirname* by creating all the non-existing parent directories first.

mkdirs creates the specified directory by using *mkdir*, as well as all nonexistent parent directories. (Effectively, *mkdirs* does the same thing as *mkdir* with the **-p** option.) The **-e** option causes *mkdirs* to list the directories it would make without actually making them. All diagnostics are those of *mkdir*. If no directories are made, *mkdir* is silent and has an exit status of 0.

EXAMPLE

To create the subdirectory structure **ltr/jd/jan**, type:

```
mkdir -p ltr/jd/jan
```

SEE ALSO

sh(1), rm(1), umask(1), intro(2), mkdir(2).

DIAGNOSTICS

mkdir returns exit code 0 if all directories given in the command line were made successfully. Otherwise, it prints a diagnostic and returns non-zero. An error code is stored in *errno*.

NAME

mkfs - construct a file system

SYNOPSIS

/etc/mkfs special [**-O**] [**-M**] blocks[:i-nodes] [gap blocks/cyl] [**-b** blocksize]

/etc/mkfs special [**-O**] [**-M**] proto [gap blocks/cyl] **-b** blocksize]

/etc/mkfs special [**-O**] [**-M**]

DESCRIPTION

The *mkfs* command constructs a file system by writing on the *special* file using the values found in the remaining arguments of the command line (except in the case of the third form of the command, discussed below). The command waits ten seconds before starting to construct the file system. During this ten-second pause the command can be aborted with a delete character.

The **-b** *blocksize* option specifies the logical block size for the file system: the number of bytes read or written by the operating system in a single I/O operation. Valid values for *blocksize* are 1024 and 4096. If the **-b** option is omitted, the default block size is 1024.

Note that if you make a 4K file system, you must add the **buffers_4k** parameter in the **dfile** to specify the number of 4K file system buffers to use.

If the second argument is a string of digits, the size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* (512-byte) disk blocks the file system occupies. If the number of i-nodes is not given, the default is the number of *logical* (1024- or 4096-byte) blocks divided by 4 (rounded down); i-nodes are allocated in groups of 16. The *mkfs* command builds a file system with a single empty directory on it. The boot program block (block zero) is left uninitialized.

If the second argument is the name of a file that can be opened, *mkfs* assumes it to be a prototype file *proto*, and takes its directions from that file. The prototype file contains tokens separated by spaces or newlines. A sample prototype specification follows (line numbers are added for clarity):

```
1       /stand/diskboot
2       4872 110
3       d--777 3 1
4       usr     d--777 3 1
5               sh      ---755 3 1 /bin/sh
6               ken     d--755 6 1
7               $
```

```
8               b0      b--644 3 1 0 0
9               c0      c--644 3 1 0 0
10              $
11      $
```

Line 1 in the example is the name of a file to be copied onto block zero as the bootstrap program.

Line 2 specifies the number of *physical* (512 byte) blocks the file system is to occupy and the number of i-nodes in the file system.

Lines 3 through 9 tell *mkfs* about files and directories to be included in this file system.

Line 3 specifies the **root** directory.

Lines 4 through 6 and 8 through 9 specify other directories and files.

The dollar sign ($) on line 7 instructs *mkfs* to end the branch of the file system it is on and continue from the next higher directory. The dollar signs on lines 10 and 11 end the process, since no additional specifications follow.

File specifications give the mode, the user ID, the group ID, and the initial contents of the file. Valid syntax for the contents field depends on the first character of the mode.

The mode for a file is specified by a six-character string. The first character specifies the type of the file. The character range is **-bcd** to specify regular, block special, character special and directory files, respectively. The second character of the mode is either **u** or **-**, to specify set-user-ID mode or not. The third character is **g** or **-**, for the set-group-ID mode. The rest of the mode is a three-digit octal number specifying the owner, group, and other read, write, and execute permissions [see *chmod*(1)].

Two decimal number tokens come after the mode; they specify the user and group IDs of the owner of the file.

If the file is a regular file, the next token of the specification can be a path name to where the contents and size are copied. If the file is a block or character special file, two decimal numbers follow, which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries . and .. and then reads a list of names and (recursively) file specifications for the entries in the directory. As noted above, the scan is terminated with the token dollar sign ($).

The first two forms of the command allow the rotational *gap* and the number of *blocks/cyl* to be specified. The default gap size is 7. The default blocks/cylinder is 400. The default is used if the supplied *gap* and *blocks/cyl* are considered illegal values, or if a short argument count occurs.

The **-O** option makes a file system with a free list instead of a bit map. This is the default for removeable disks.

The **-M** option makes a file system with a bit map in addition to a free list. This is the default for fixed disks.

Special must be a disk slice. The third form of the *mkfs* command extracts the slice size from the Volume Home Block and creates a file system the same size; this third option cannot be used where there are overlapping partitions. The number of i-nodes is the number of logical blocks divided by 4. Optimal values for gap size and blocks/cylinder are calculated; these may not be 7 and 400.

SEE ALSO

chmod(1), dir(4), fs(4).
S/Series CTIX Administrator's Guide.

BUGS

With a prototype file, there is no way to specify links. The maximum number of i-nodes configurable is 65500.

NAME

mkhosts - make node name commands

SYNOPSIS

/etc/mkhosts

DESCRIPTION

mkhosts makes the simplified forms of the *rcmd*(1) and *rlogin*(1) commands. For each node listed in **/etc/hosts**, *mkhosts* creates a link to **/usr/local/bin/rcmd** in **/usr/hosts**. Each link's name is the same as the node's official name in **/etc/hosts**.

SEE ALSO

rcmd (1), rlogin(1).

NAME

mkifile - make an ifile from an object file

SYNOPSIS

/etc/lddrv/mkifile a.out ifile

DESCRIPTION

mkifile takes an object module and writes in *ifile* a line of the form

symbol = 0x*value*

For each external symbol in the object module this ifile can be used as an argument to *ld*(1) as an absolute symbol table against which other modules may be linked. *mkifile* is used with loadable drivers to provide the symbols for the currently running CTIX.

SEE ALSO

ld(1), lddrv(1M), ldeeprom(1M).

NAME

mklost+found - make a lost+found directory for fsck

SYNOPSIS

/etc/mklost+found

DESCRIPTION

A directory *lost+found* is created in the current directory and a number of empty files are created therein and then removed so that there will be empty slots for *fsck*(M). This command should be run immediately after first mounting a newly created file system.

SEE ALSO

fsck(1M), mkfs(1M).
S/Series CTIX Administrator's Guide.

BUGS

Should be done automatically by *mkfs*.

WARNING

It is dangerous to run *mklost+found* if the free list is not clean.

NAME

mknod - build special file

SYNOPSIS

/etc/mknod name **b** | **c** major minor

/etc/mknod name **p**

DESCRIPTION

The *mknod* command makes a directory entry and corresponding i-node for a special file.

The first argument is the *name* of the entry. The convention is to keep such files in the **/dev** directory.

In the first case, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (that is, unit, drive, or line number). The major and minor numbers can be specified in decimal or octal. The assignment of major device numbers is specific to each system. The information is contained in the system master file **/etc/master**. You must be the superuser to use this form of the command.

The second case is the form of the *mknod* used to create FIFO's (also known as *named pipes*).

WARNING

If *mknod* is used to create a device in a remote directory (Remote File Sharing), the major and minor device numbers are interpreted by the server.

SEE ALSO

createdev(1M), mknod(2).

NAME

mkshlib - create a shared library

SYNOPSIS

mkshlib **-s** specfil [**-t** target] [**-h** host] [**-n**] [**-L** dir ...] [**-q**]

DESCRIPTION

The *mkshlib* command builds both the host and target shared libraries. A shared library is similar in function to a normal, non-shared library, except that programs which link with a shared library will share the library code during execution whereas programs which link with a non-shared library will get their own copy of each library routine used.

The host shared library is an archive which is used to link-edit user programs with the shared library [see *ar*(4)]. A host shared library can be treated exactly like a non-shared library and should be included on *cc*(1) command lines in the usual way [see *cc*(1)]. Further, all operations which can be performed on an archive can also be performed on the host shared library.

The target shared library is an executable module that is bound into the user's address space during execution of a program using the shared library. The target shared library contains the code for all the routines in the library and must be fully resolved. The target will be brought into memory during execution of a program using the shared library, and subsequent processes that use the shared library will share the copy of code already in memory. The text of the target is always shared, but each process will get its own copy of the data.

The user interface to *mkshlib* consists of command line options and a shared library specification file. The shared library specification file describes the contents of the shared library.

The *mkshlib* command invokes other tools such as the archiver, *ar*(1), the assembler, *as*(1), and the loader, *ld*(1). Tools are invoked through the use of *execvp* [see *exec*(2)], which searches directories in the user's PATH. Also, prefixes to *mkshlib* are parsed in the same manner as prefixes to the *cc*(1) command, and invoked tools are given the prefix, where appropriate. For example, *pfxmkshlib* will invoke *pfxld*.

The following command line options are recognized by *mkshlib*:

-s specfil — Specifies the shared library specification file, *specfil*. This file contains the information necessary to build a shared library.

-t target — Specifies the output filename of the target shared library being created. It is assumed that this file will be installed on the target machine at the location given in the specification file (see the

#target directive below). If the **-n** option is used, then a new target shared library will not be generated.

-h host — Specifies the output filename of the host shared library being created. If this option is not given, then the host shared library will not be produced.

-n — Do not generate a new target shared library. This option is useful when producing only a new host shared library. The **-t** option must still be supplied since a version of the target shared library is needed to build the host shared library.

-L *dir* ... — Change the algorithm of searching for the host shared libraries specified with the **#objects noload** directive to look in *dir* before looking in the default directories. The **-L** option can be specified multiple times on the command line in which case the directories given with the **-L** options are searched in the order given on the command line before the default directories.

-q — Quiet warning messages. This option is useful when warning messages are expected but not desired.

The shared library specification file contains all the information necessary to build both the host and target shared libraries. The contents and format of the specification file are given by the directives listed below.

All directives that can be followed by multi-line specifications are valid until the next directive or the end of the file.

#address sectname address

Specifies the start address, *address*, of section *sectname* for the target. This directive typically is used to specify the start addresses of the .text and .data sections. One **#address** per section name is valid. A **#address** directive must be given exactly once for the **.text** section and once for the **.data** section.

#target pathname

Specifies the absolute pathname, *pathname*, at which the target shared library will be installed on the target machine. The operating system uses this pathname to locate the shared library when executing **a.out** files that use this shared library. This directive must be specified exactly once per specification file.

#branch

Specifies the start of the branch table specifications. The lines following this directive are taken to be branch table specification lines.

Branch table specification lines have the following format:

funcname <white space> position

where *funcname* is the name of the symbol given a branch table entry and *position* specifies the position of *funcname*'s branch table entry. *Position* may be a single integer or a range of integers of the form *position1-position2*. Each *position* must be greater than or equal to one, the same position can not be specified more than once, and every position from one to the highest given position must be accounted for.

If a symbol is given more than one branch table entry by associating a range of positions with the symbol or by specifying the same symbol on more than one branch table specification line, then the symbol is defined to have the address of the highest associated branch table entry. All other branch table entries for the symbol can be thought of as "empty" slots and can be replaced by new entries in future versions of the shared library.

Finally, only functions should be given branch table entries, and those functions must be external symbols.

This directive must be specified exactly once per shared library specification file.

#objects

The lines following this directive are taken to be the list of input object files in the order they are to be loaded into the target. The list simply consists of each pathname followed by a newline character. This list is also used to determine the input object files for the host shared library, but the order for the host is given by running the list through *lorder*(1) and *tsort*(1).

This directive must be specified exactly once per shared library specification file.

#objects noload

The **#objects noload** is followed by a list of host shared libraries. These libraries are searched in the order listed to resolve undefined symbols from the library being built. During the search it is considered an error if a non-shared version of a symbol is found before a shared version of the symbol.

Each name given is assumed to be a pathname to a host or an argument of the form **-lX**where **libX.a** is the name of a file in LIBDIR or LLIBDIR. This behavior is identical to that of *ld* , and the **-L** option can

be used on the command line to specify other directories in which to locate these archives.

Note that if a host shared library is specified using **#objects noload**, any *cc* command that links to the shared library being built will need to specify that host also.

#hide linker [*]

This directive changes symbols that are normally **external** into **static** symbols, local to the library being created. A regular expression may be given [*sh*(1), *find*(1)], in which case all **external** symbols matching the regular expression are hidden; the **#export** directive (see below) can be used to counter this effect for specified symbols.

The optional "*" is equivalent to the directive

```
#hide linker
  *
```

and causes all **external** symbols to be made into **static** symbols.

All symbols specified in **#init** and **#branch** directives are assumed to be **external** symbols, and cannot be changed into **static** symbols using the **#hide** directive.

#export linker [*]

Symbols given in the **#export** directive are **external** symbols (global among files) that, because of a regular expression in a **#hide** directive, would otherwise have been made static. For example,

```
#hide linker *
#export linker
  one
  two
```

causes all symbols except *one*, *two*, and those used in **#branch** and **#init** entries to be tagged as **static**.

#init object

Specifies that the object file, *object*, requires initialization code. The lines following this directive are taken to be initialization specification lines.

Initialization specification lines have the following format:

pimport <white space> import

Pimport is a pointer to the associated imported symbol, *import*, and must be defined in the current specified object file, *object*. The initialization code generated for each such line is of the form:

```
pimport = &import;
```

All initializations for a particular object file must be given at once and multiple specifications of the same object file are not allowed.

#ident string
Specifies a string, *string*, to be included in the .comment section of the target shared library. This directive can be specified only once

Specifies a comment. All information on a line following this directive is ignored.

FILES

*TEMPDIR/**	temporary files

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

LIBDIR	usually /lib
LLIBDIR	usually /usr/lib

SEE ALSO

ar(1), as(1), cc(1), chkshlib(1), ld(1), lorder(1), tsort(1), a.out(4), ar(4).
UNIX System V Release 3.2 Programmer's Guide.

CAVEATS

The **-n** option cannot be used with the **#objects noload** directive.

If *mkshlib* is asked to create a host library and a host of that name already exists, *mkshlib* will update the host using **ar -ru**. This means that you should always remove the host before rebuilding whenever an object file previously included in the library is removed or renamed.

If the address specified with the **#address** directive is outside user space, the library build may look successful, but if you try to use it, it might not work.

NAME

mktpy, mvtpy - install or relocate a PT or GT local printer

SYNOPSIS

/usr/local/bin/mktpy [name1 [name2 [tty]]]
/usr/local/bin/mvtpy name tty n

DESCRIPTION

To install a PT or GT local printer initially, the system administrator must use the *lpadmin*(1M) command. The administrator *must* use the **-l** flag to ensure that the spooling system knows that the printer is attached to a login terminal. If this is not done, the printers will be attached to random devices at boot time, and behavior is undefined.

mktpy is used to inform the *lp*(1) spooling system of the location of a printer that is attached to a PT or GT and to *enable*(1) the printer to accept print requests. *mktpy* installs a printer on an RS-422 terminal's local port. *mvtpy* updates a *mktpy* installation by changing the device association when the terminal's device number changes. The device number may change each time the system or the terminal is powered off and on.

The *mktpy* command accepts the names of one or two printers as arguments. If no arguments are given, it will prompt for the name of two printers. As an argument, the null string "" must be used in the first argument position if only one printer exists and is attached to the second port. In response to a prompt, a <cr> indicates that no printer is attached to the indicated port.

If *mktpy* is run and the indicated printer is already attached to another tty device, the location of that printer is not changed and a warning message is printed.

If the command is being executed to inform the system of a printer attached to a terminal other than the terminal from which the command is being run, then the printers must be specified as arguments (with a null string, if necessary, as a place holder). The tty line to which the printer's host terminal is attached must be specified as the third argument.

The *mvtpy* command is most commonly used when a PT or GT has been powered off and then on again and the tty number of the PT or GT has changed. The system needs to be informed of the tty line that should receive the print requests. Before running *mvtpy*, the new tty number must be determined. *Name1* is the name of the printer, *name2* is the tty number of the new tty, and *n* is the port on the PT or GT to which the printer is attached; *n* must be either 1 or

2. *mvtpy* processes these arguments and issues an *lpadmin*(1M) command to move the specified printer to the indicated tty line. Diagnostics are from *lpadmin*.

For the convenience of users, a **mktpy** login is provided. The *mktpy* login executes *mktpy* for the terminal where it is executed and prompts for printer names.

EXAMPLES

Use the following command to install two printers on the same tty:

mktpy DIABLO EPSON

The following command installs a printer on the second port of another tty:

mktpy '' QUME tty256

The following command tells the system that the tty number of the terminal to which a printer is attached has changed:

mvtpy QUME tty260

NOTES

Only PT or GT terminals that are on cluster lines can have printers attached to the serial ports. GT terminals have two serial ports and PT terminal s have one serial port. Printers attached to PT terminals are considered to be attached to the first port. For each terminal **/dev/tty***nnn* on the system, two other character special files exist, **/dev/tp/a***nnn* and **/dev/tp/b***nnn*. These devices refer to the first and second serial ports respectively on the matching terminal.

FILES

/dev/tty*nnn*	Name of tty
/dev/tpa	First serial port of /dev/tty*nnn*
/dev/tpb	Second serial port of /dev/tty*nnn*
/tmp/mkt[12]*	Temporary files to hold printer status for mktpy
/tmp/mvt*	Temporary file to hold printer status for mvtpy
/dev/tp/*	

SEE ALSO

accept(1), disable(1), enable(1), lp(1), lpadmin(1M), lpsched(1M), reject(1), tp(7).
S/Series CTIX Administrator's Guide.

DIAGNOSTICS

Generally self explanatory. "LP doesn't know *xxxx*" if the printer has not been created yet. Check spelling of printer name or see system administrator.

WARNINGS

The following command has no effect; it does not prompt for input:

mktpy " " ttyxxx

Printing to the second port of a PT is not recommended (this is because it is addressable even though it does not exist).

XOFF from the printer can not be distinguished from XOFF from the keyboard of the terminal controlling the printer: thus, the printer must not use flow control.

NAME

mm, checkmm - print/check documents formatted with the MM macros

SYNOPSIS

mm [options] [files]

checkmm [files]

DESCRIPTION

The *mm* command can be used to print documents using *nroff* and the MM text-formatting macro package. The command has options to specify preprocessing by *tbl*(1) and/or *neqn* [see *neqn*(1)] and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for *nroff* and MM are generated, depending on the options selected.

Options for *mm* are listed and described below. Any other arguments or flags (for example, **-rC3**) are passed to *nroff* or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

-T*term* Specifies the type of output terminal. (Use **help term2**. to list recognized values for *term*.) If **-T** is *not* specified, *mm* uses the value of the shell variable **$TERM** from the environment [see *profile*(4) and *environ*(5)] as the value of *term*, if **$TERM** is set; otherwise, *mm* uses **450** as the value of *term*. If several terminal types are specified, the last one takes precedence.

-12 Indicates that the document is to be produced in 12-pitch. Can be used when **$TERM** is set to one of **300**, **300s**, **450**, and **1620**. (The pitch switch on the DASI 300 and 300s terminals must be manually set to **12** if this option is used.)

-c Causes *mm* to invoke *col*(1); note that *col*(1) is invoked automatically by *mm* unless *term* is one of **300**, **300s**, **450**, **37**, **4000a**, **382**, **4014**, **tek**, **1620**, and **X**.

-e Causes *mm* to invoke *neqn*; also causes *neqn* to read the **/usr/pub/eqnchar** file [see *eqnchar*(5)].

-t Causes *mm* to invoke *tbl*(1).

-E Invokes the **-e** option of *nroff*.

-y Causes *mm* to use the non-compacted version of the macros [see *mm*(5)].

As an example (assuming that the shell variable **$TERM** is set in the environment to **450**), the two command lines below are equivalent:

mm -t -rC3 -12 ghh*
tbl ghh* | nroff -cm -T450-12 -h -rC3

The *mm* command reads the standard input when - is specified instead of file names. (Mentioning other files together with - leads to disaster.) This option allows you to use *mm* as a filter; for example:

cat dws | mm -

The *checkmm* program checks the contents of the named *files* for errors in the use of the Memorandum Macros, missing or unbalanced *neqn* delimiters, and **.EQ/.EN** pairs. Note that you need not use the *checkeq* program [see *neqn*(1)]. Appropriate messages are produced. The program skips all directories, and if no file name is given, standard input is read.

HINTS

The *mm* command invokes *nroff* with the **-h** flag. With this flag, *nroff* assumes that the terminal has tabs set every eight character positions.

Use the **-o***list* option of *nroff* to specify ranges of pages to be output. Note, however, that *mm*, if invoked with one or more of the **-e**, **-t**, and - options, *together* with the **-o***list* option of *nroff* can cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

If you use the **-s** option of *nroff* (to stop between pages of output), use line-feed (rather than return or newline) to restart the output. The **-s** option of *nroff* does not work with the **-c** option of *mm*, or if *mm* automatically invokes *col*(1) (see the description of the **-c** option above).

If you specify a different terminal than the actual output terminal, *mm* produces (often subtle) garbage; however, if you are redirecting output into a file, use the **-T37** option, and then use the appropriate terminal filter when you actually print that file.

SEE ALSO

col(1), env(1), neqn(1), greek(1), nroff(1), tbl(1), profile(4), mm(5), term(5).

DIAGNOSTICS

The *mm* command can produce the following diagnostic message:

mm: no input file
None of the arguments is a readable file, and *mm* is not used as a filter.

The *checkmm* command can produce the following diagnostic message:

Cannot open filename

The file(s) is unreadable. The remaining output of the program is diagnostic of the source file.

MMT(1)

NAME

mmt, mvt - typeset documents, view graphs, and slides

SYNOPSIS

mmt [options] [files]

mvt [options] [files]

DESCRIPTION

These two commands are very similar to *mm*(1), except that they both typeset their input via *troff*(1), as opposed to formatting it via *nroff*; *mmt* uses the MM macro package, while *mvt* uses the Macro Package for View Graphs and Slides. These two commands have options to specify preprocessing by *tbl*(1) and/or *eqn*(1). The proper pipelines and the required arguments and flags for *troff*(1) and for the macro packages are generated, depending on the options selected.

Options are given below. Any other arguments or flags (for example, **-rC3**) are passed to *troff*(1) or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, these commands print a list of their options.

-e Causes these commands to invoke *eqn*(1); also causes *eqn* to read the **/usr/pub/eqnchar** file [see *eqnchar*(5)].

-t Causes these commands to invoke *tbl*(1).

-Tvp Directs the output to a Versatec printer; this option is not available at all sites.

-T4014 Directs the output to a Tektronix 4014 terminal via the *tc*(1) filter.

-Ttek Same as **-T4014**.

-a Invokes the **-a** option of *troff*(1).

-y Causes *mmt* to use the non-compacted version of the macros [see *mm*(5)]. No effect for *mvt*.

These commands read the standard input when - is specified instead of any file names.

mvt is just a link to *mmt*.

HINT

Use the **-o***list* option of *troff*(1) to specify ranges of pages to be output. Note, however, that these commands, if invoked with one or more of the **-e**, **-t**, and **-** options, *together* with the **-o***list* option of *troff*(1) may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

SEE ALSO

env(1), eqn(1), mm(1), tbl(1), tc(1), troff(1), profile(4), environ(5), mm(5), mv(5).

DIAGNOSTICS

"m[mv]t: no input file" if none of the arguments is a readable file and the command is not used as a filter.

NAME

more, page - text perusal

SYNOPSIS

more [**-cdflsu**] [*-n*] [*+linenumber*] [*+/pattern*] [name ...]

page [**-cdflsu**] [*-n*] [*+l*] [*+/pattern*] [name ...]

DESCRIPTION

more and *page* display text a screenful at a time. *page* clears the screen before each screenful; otherwise, *page* and *more* are identical. The rest of this description uses *more*.

When the screen is full, *more* prints the string "--More--". If input is a file, *more* indicates how much of the file has been read. To display the next screenful, type a space. To display a list of commands, type an "h".

more treats underlining and form feeds (^L) specially; otherwise it passes along its input unmodified. If your terminal has an underline mode or some other standout mode, *more* uses this mode to display underlined text. If a file begins with a form feed, *more* clears the screen before displaying the file. Subsequent form feeds cause *more* to pause.

If the standard output is not a terminal, *more* does not pause between screenfuls.

To make *more* pause in the middle of a screenful, type QUIT (normally code-\). Some text is lost when you do this, and you may terminate whatever program is piping to *more*.

These are the options.

-*n* Display *n* lines instead of a screenful.

-c Avoid scrolling. *more* begins each screenful at the top of the screen and erases each line as it is needed.

-d Prompts user with "Hit space to continue, Rubout to abort" at the end of each screenful.

-f Count file lines, rather than screen lines.

-l No special treatment for form feeds.

-s Suppress all but one of each sequence of blank lines. Useful with *nroff*.

-u No special handling of underlining.

+linenumber

Begin displaying the file at line number *l*.

+/pattern

Search for *pattern* and begin displaying the file two lines before it. *Pattern* is a regular expression; it follows the same rules as does a search in *ex*(1).

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.

To supply options automatically, put the options in the **MORE** environment variable. Here's an example for the Bourne Shell:

MORE = '-s -d'

more looks at the **TERM** environment variable to find what kind of terminal you're using and at the **TERMCAP** environment variable to find how the terminal works. If **TERMCAP** is not set, *more* examines */etc/termcap*.

more resets terminal modes. This permits character (as opposed to line) commands and limits echoing of commands.

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 that 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 that 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 (code-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 quotation mark) 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; that is, it is not necessary to type a carriage return. Up to the time when the command character itself is given, the

user can press the line kill character to cancel the numerical argument being formed. In addition, the user can press the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can press the quit key (normally code-\). *more* will stop sending output, and will display the usual --More-- prompt. The user can then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done because 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 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
```

FILES

/etc/termcap	Terminal database
/usr/lib/more.help	Help file

SEE ALSO

csh(1), man(1), script(1), sh(1), termcap(4), environ(5).

NAME

mount, umount - mount and unmount file systems and remote resources

SYNOPSIS

/etc/mount

/etc/mount [**-r**] [**-f** fstyp] special directory

/etc/mount [**-r**] **-f NFS** [,options] special directory

/etc/mount [**-r**] [**-c**] **-d** resource directory

/etc/umount special

/etc/umount [**-d**] resource

DESCRIPTION

File systems other than **root** (/) are considered *removable* in the sense that they can be either available to users or unavailable. The *mount* command announces to the system that *special* (a block special device) or *resource* (a remote resource) is available to users from the mount point *directory*. Note that *directory* must already exist; it becomes the name of the root of the newly mounted *special* or *resource*. A unique resource can be mounted only once (no multiple mounts).

When invoked with no arguments, *mount* displays the entire mount table. When entered with arguments, *mount* adds an entry to the table of mounted devices, **/etc/mnttab**. The *umount* command removes the entry. If invoked with any of the following partial argument lists, *mount* searches **/etc/fstab** for the missing arguments: *special*, **-d** *resource*, *directory*, or **-d** *directory*.

The following options are available:

-v With no other arguments, prints a more verbose mount table containing file system type identifier (S51K, DUFST, NFS); with other arguments, print the fully expanded *mount* command before mounting.

-r Indicates that *special* or *resource* is to be mounted read-only. If *special* or *resource* is write-protected or read-only advertised, this flag must be used.

-c Indicates that remote reads and writes should not be cached in the local buffer pool. **-c** is used in conjunction with **-d**.

-d Indicates that *resource* is a remote resource to be mounted on *directory* or unmounted. To mount a remote resource, Remote File Sharing (RFS) or the Network File System (NFS) must be up and running, and the resource must be advertised by a remote computer

[see *rfstart*(1M) and *adv*(1M)]. If **-d** is not used, and this is not an NFS mount, *special* must be a local block special device.

-f *fstyp* Indicates that *fstyp* is the file system type to be mounted. If this argument is omitted, it defaults to the **root** *fstyp*.

If *fstyp* is **NFS**, NFS comma-separated options can be added after the *fstyp*. The available NFS options follow:

soft Return error if the server doesn't respond.

rsize=*n* Set the read-buffer size to *n* bytes.

wsize=*n* Set the write-buffer size to *n* bytes.

timeo=*n* Set the initial NFS timeout to *n* tenths of a second.

retrans=*n* Set the number of NFS retransmissions to *n*.

port=*n* Set the server IP port number to *n*.

special Indicates the block special device to be mounted on *directory*. If *fstyp* is **NFS**, *special* should be of the form *hostname*:*/pathname*.

resource Indicates the remote resource name to be mounted on *directory*.

directory Indicates the directory mount point for *special* or *resource*. (The directory must already exist.)

The *umount* command announces to the system that the file system previously mounted *special* or *resource* is to be made unavailable. If invoked with an incomplete argument list, *umount* searches **/etc/fstab** for the missing arguments.

Note that *mount* can be used by any user to list mounted file systems and resources. Only the super-user can mount and unmount file systems.

FILES

/etc/mnttab mount table
/etc/fstab file system table

SEE ALSO

adv(1M), fuser(1M), mountd(1M), nfsd(1M), nsquery(1M), rfstart(1M), rmntstat(1M), setmnt(1M), showmount(1M), unadv(1M), mount(2), umount(2), fstab(4), mnttab(4).
S/Series CTIX Administrator's Guide.

DIAGNOSTICS

If the *mount*(2) system call fails, *mount* prints an appropriate diagnostic. The *mount* command issues a warning if the file system to be mounted is currently mounted under another name. A remote resource mount fails if the resource is not available, or if it is advertised read-only and not mounted with **-r**, or if Remote File Sharing is not running.

The *umount* command fails if *special* or *resource* is not mounted or if it is busy. *special* or *resource* is busy if it contains an open file or some user's working directory. In such a case, use *fuser*(1M) to list and kill processes using *special* or *resource*.

WARNINGS

Physically removing a mounted file system diskette from the diskette drive before issuing the *umount* command damages the file system.

NAME

mountall, umountall - mount, unmount multiple file systems

SYNOPSIS

/etc/mountall [-] [file-system-table] ...

/etc/umountall [**-k**]

DESCRIPTION

mountall is used to mount local file systems according to a *file-system-table.* (**/etc/fstab** is the default file system table.) The special file name "-" reads from the standard input. All remote RFS file systems can be mounted using *rmountall(1m)* and remote NFS file systems mounted using *nmountall(1m)* .

Before each file system is mounted, it is checked using *fsstat*(1M) to see if it appears mountable. If the file system does not appear mountable, it is checked, using *fsck*(1M), before the mount is attempted.

umountall causes all mounted file systems, both local and remote, except **root** to be unmounted. The **-k** option sends a SIGKILL signal, via *fuser*(1M), to processes that have files open. Although *umountall* handles remote file systems, *rumountall* should be used for RFS and *numountall* used for NFS.

These commands may be executed only by the super-user.

FILES

File-system-table format:

column 1	Block special file name of file system.
column 2	Mount-point directory.
column 3	**-r** if to be mounted read-only; **-d** if remote.
column 4	(Optional) file system type string.
column 5+	Ignored.

White-space separates columns. Lines beginning with a pound sign (#) are comments. Empty lines are ignored.

A typical file-system-table might read:

```
/dev/dsk/c1d0s2 /usr -r S51K
```

SEE ALSO

fsck(1M), fsstat(1M), fuser(1M), mount(1M), signal(2).

DIAGNOSTICS

No messages are printed if the file systems are mountable and clean.

Error and warning messages come from *fsck*(1M), *fsstat*(1M), and *mount*(1M).

BUGS

Care should be taken when using with remote file systems since the network could be down or the host may be in or going to single-user mode where the network has been dismantled.

NAME

mountd - NFS mount request server

SYNOPSIS

/etc/mountd

DESCRIPTION

mountd is an *rpc* server that answers file system mount and exported file system information requests. It reads the file **/etc/exports**, described in *exports*(4), to determine which file systems are available to which machines and users. It also provides information as to which clients have file systems mounted. This information can be printed using the *showmount*(1M) command.

mountd is started automatically when the system is booted if there is a zero-length file **/etc/rcopts/KNFS**.

FILES

/etc/rc[23].d/S95nfs
/etc/rcopts/KNFS

SEE ALSO

exports(4), services(4), showmount(1M).

NAME

mvdir - move a directory

SYNOPSIS

/etc/mvdir dirname name

DESCRIPTION

mvdir moves directories within a file system. *Dirname* must be a directory. If *name* does not exist, it will be created as a directory. If *name* does exist, *dirname* will be created as *name/dirname*. *Dirname* and *name* may not be on the same path; that is, one may not be subordinate to the other. For example:

```
mvdir x/y x/z
```

is legal, but

```
mvdir x/y x/y/z
```

is not.

SEE ALSO

mkdir(1), mv(1).

WARNINGS

Only the super-user can use *mvdir*.

NAME

named - Internet domain name server

SYNOPSIS

named [**-p** port] [bootfile]

DESCRIPTION

The *named* (Berkeley Internet Name Domain, or BIND) server for distributed database systems implements DARPA Internet domains. Its chief use is storing and retrieving host names and addresses; therefore, it can be used as a replacement for **/etc/hosts** [see *hosts*(4)]. Both *named* and the original host lookup mechanism are supported under CTIX Internetworking.

The domain name server's great advantage over the original host lookup table (**/etc/hosts**) and the *gethosts*(1M) program is that it makes it unnecessary to propagate copies of a single, master database for host name resolution over an entire network. The original host lookup mechanism works well for a small, single-domain network where a master database is easily maintained and propagated from a central location. The advantage of *named* is seen in a large, multiple-domain network where machines cross organizational boundaries. Thus, a network configuration using *named* is appropriate in the following network situations:

- Where the configuration changes often (say, once a week).
- Where it is difficult to maintain centralized control over the network.
- Where DARPA RFC compliance is required.

Options are interpreted as follows:

-p *port* Use *port* rather than the port specified in **/etc/services**.

bootfile Use *bootfile* rather than **/usr/local/domain/named.boot** (the default) for *named*'s boot-time information.

The Domain Name Space

The *named* server allows the entire name space to be partitioned into a tree-like structure of domains. A *domain* is a basic indivisible entity containing a set of hosts administered by the same authority.

The domain name is a dot-separated list of the domains from the current domain to the root, ordered from left to right. (Each domain name above the current domain in the dot-separated list is given as the left-most qualifier of its own full domain name. For example, if the nodes in the tree from the current node to the root are **Sales**, **MySite**, and **COM**, the domain name is **Sales.MySite.COM**, not **Sales.MySite.COM.MySite.COM.COM.**) The host name consists of

the domain name with an additional qualifier, signifying the host, appended to the left side: for example, **jack.Sales.MySite.COM**. A host name need only be unique within the current domain.

Conceptually, each domain names a set of information, and query operations are attempts to extract specific types of information from a particular set. A query specifies the domain of interest and the type of resource information that is desired. For example, the ARPA Internet uses some of its domain names to identify hosts; queries for address resources return ARPA Internet host addresses. Note that domain names are not required to have a one-to-one correspondence with host names, host addresses, or any other type of information.

Note that the the term *domain* has a different meaning in the Internet name server context from its meaning in the Remote File Sharing context. For an explanation of domain as it is used in connection with Remote File Sharing, see the *S/Series CTIX Administrator's Guide*.

Name Server Functions and Types

Name servers are programs that hold information about the domain's tree structure and associated information (addresses, resources, and so forth). A name server may cache information about any part of the domain tree, but in general a particular name server has complete information about a subset of the domain space, and pointers to other name servers that can lead to information about any part of the domain tree. Name servers know the parts of the domain tree for which they have complete information: these parts are called *zones*. A name server is an *authority* for these parts of the name space. A zone can be as small as a single domain or as large as an extensive subhierarchy of the total domain name space. The important distinction here is that a zone is the authoritative unit for a particular server. In effect, the division of the name space into zones allows the grouping of domains into larger units for the purpose of reducing the number of authoritative servers. A zone is named after the highest level domain in it.

There are two basic types of servers:

1 *Master* server. A master server is an authority for a zone. A *primary master* maintains the database for its zone of authority on disk. There can be multiple primary masters for a zone, but the benefits of such a configuration must be weighed against the risk of introducing inconsistencies into the database.

In addition to a primary master, each zone should have at least one *secondary master*. A secondary master is also an authority for the zone. Its authority is delegated by the primary master, and it receives

its data from the primary: At boot time a secondary master requests all the data for the zone; the data is loaded into memory, and the secondary server then periodically checks with the primary server to see if it needs to update its cache. (The frequency of periodic refresh checks is configurable, as described below in the "Standard Resource Record Format" description.)

A master server can be primary for one zone and secondary for another.

The concept of primary and secondary servers applies also to the special domain, IN-ADDR.ARPA, discussed below.

2 *Caching Only* server. Any machine that actually runs *named* locally is running a caching server, since the server caches the information it receives until the data expires (data expiration is determined by the *time_to_live* field attached to the data when it is received).

A *Caching Only* server is a server that is not authoritative for any part of the name space but services queries and asks other servers who have the authority for the information needed.

A host does not need to run the name server to run all the networking programs that use the name server: a workstation or other machine with limited memory and CPU can be a *serverless node*. This option allows all queries to be serviced by a remote name server (that is, a name server running on another machine on the network).

Resolvers

Resolvers are routines that extract information from name servers in response to user program requests: these routines are used specifically for making, sending, and interpreting packets to Internet domain name servers [see *resolver*(3)]. Resolvers must be able to access at least one name server and use that name server's information to answer a query directly, or pursue the query using referrals to other name servers.

The IN-ADDR.ARPA Domain

User programs extract information from servers with either a name or an address as a key. Since Internet addresses are organized as a hierarchy that is separate from the host name space hierarchy (which is the fundamental domain hierarchy), name server operations need a mechanism for mapping between the two hierarchies. This mechanism is provided by *inverse mapping*, and it uses the **IN-ADDR.ARPA** domain as a special domain for mapping addresses to names. An Internet address suffixed by a **IN-ADDR.ARPA** label specifies the reversal of the address for inverse mapping purposes and signifies its location in

the IN-ADDR.ARPA domain. For example, the Internet address **45.0.0.7** is located in the domain **7.0.0.45.IN-ADDR.ARPA**. Note that all four octets must be specified, even if an octet is zero.

IN-ADDR.ARPA domains have primary and secondary servers that are authoritative for these special domains.

Inverse mapping is discussed below in relation to the **named.boot** file, the **hosts.rev** file, and **PTR** record type.

Configuration Files

named uses several configuration files to load its data. Any machine running a server locally must configure each type of file except for the *hosts* file, which is configured only on a primary server, and the *IN-ADDR.ARPA* database file. The set of entries for a file depends on whether the server is a primary server, a secondary server, or a caching only server. A remote server configuration is a special case: a serverless node needs only one configuration file, **/usr/local/domain/resolv.conf**.

The boot file and the resolver file (**resolv.conf**) have their own formats for data. All other files use the "Standard Resource Record Data" format, which is described in a separate subsection below.

Discussions that follow refer to a sample configuration in which **MySite.COM** is the root domain (**MySite** is not on the DARPA Internet in this example); there are four subdomains under the root domain (**Sales**, **Marketing**, **Finance**, and **Training**); and there are also hosts (leaves) in a corporate unit in the root domain. Any of the four subdomains could have subordinate subdomains, but in this example they do not. Complete examples are given under EXAMPLES below.

Boot File

Each machine running the name server reads its boot file when it first comes up. By default, the name of the boot file is **/usr/local/domain/named.boot**, but a different file name can be specified as a parameter to *named*. The boot file tells the server what type of server it is, which zones it has authority over, and where to get its initial data.

Entries in the boot file are as follows:

Domain:
The line in the boot file that specifies the default domain consists of two fields: the keyword **domain** and the domain name, as shown below:

```
domain Sales.MySite.COM
```

When the name server receives a query for a name without a dot (.), it appends the domain name string, as specified in this entry, to the name. The *domain* entry is specified in every boot file. If more than one domain entry is specified, each domain is tried in the order given.

Sortlist:
The line in the boot file that specifies a preferential order for sorting addresses (in the case of a machine that has multiple interfaces) consists of two fields: the keyword **sortlist** and a list of the network numbers in order of priority, starting with the highest priority, as shown below:

```
sortlist 3.0.0.0 128.1.0.0
```

This entry is required only if you want to control the order that addresses are returned from the name server.

Primary Master:
The line in the boot file that specifies that the current host is a primary master for a domain or zone consists of three fields: the keyword **primary**, the domain name for which it is authoritative, and the file from which host data is read (by convention, **/usr/local/domain/named.hosts**), as shown below:

```
primary Sales.MySite.COM /usr/local/domain/named.hosts
```

This entry appears only in the boot file for a server that is a primary master for a domain or zone.

Secondary Master:
The line in the file that specifies that the current host is a secondary master for a domain or zone consists of three fields: the keyword **secondary**, the domain name for which it is authoritative, and a list of the network addresses for the name servers that are primary for the domain or zone, as shown below:

```
secondary Sales.MySite.COM 3.0.0.24 3.0.0.94
```

Addresses are tried in successive order. (In the above example, the machine that has this boot file might have the Internet domain name **jack.Sales.MySite.COM**.)

Secondary entries can also list local disk files containing backup information to be used in the event that none of the specified addresses can be reached. *named* periodically dumps its information into such a backup file: if a disk file is used by the secondary server when it boots, the server gets refreshes when one of the addresses becomes reachable. For example, the following entry specifies that *named* should dump the hosts file to

/usr/local/domain/hosts.bak and should use this file if it cannot reach either of the two authoritative servers to obtain its zone information:

```
secondary    Sales.MySite.COM 3.0.0.24    3.0.0.94
                /usr/local/domain/hosts.bak
```

This entry appears only in a boot file for a server that is a secondary master for a domain or zone.

Cache:
The line in the boot file that specifies the file containing the server's initial cache information (to prime the cache) consists of three fields: the keyword **cache**, a dot (.) (which denotes the current machine), and the file name of the cache file (by convention, **/usr/local/domain/named.ca**), as shown below:

```
cache  .    /usr/local/domain/named.ca
```

This entry is specified in every boot file. What distinguishes a boot file for a *Caching Only* server is the absence of primary and secondary master entries, except for the loopback entry described immediately below.

Local Loopback:
The line in the file that specifies that the current host is a primary server for its own loopback address consists of three fields: the keyword **primary**, the address of the loopback in the **IN-ADDR.ARPA** domain (the address **0.0.127.IN-ADDR.ARPA**), and the name of the *localhost* file (by convention, **/usr/local/domain/named.local**), as shown below:

```
primary  0.0.127.IN-ADDR.ARPA    /usr/local/domain/named.local
```

This entry is specified in every boot file.

Primary/Secondary Master for IN-ADDR.ARPA Domain:
IN-ADDR.ARPA domains, used for host information lookups where the address rather than the host name is given as a key, require primary and secondary master servers. An entry for a primary server consists of three fields: the keyword **primary**, the IN-ADDR.ARPA domain, and the name of the file containing the reverse lookup database for this IN-ADDR.ARPA domain, as shown below:

```
primary  0.0.3.IN-ADDR.ARPA    /usr/local/domain/hosts.rev
```

An entry for a secondary server specifies **secondary** instead of **primary**, the IN-ADDR.ARPA domain, and one or more addresses of primary IN-ADDR.ARPA servers. A backup file may also be specified (see the discussion above about secondary entries with backup file specifications).

Cache Initialization File

Each machine running the name server primes its cache with information about the *root* authoritative servers for the network with data specified in the cache initialization file. (By convention, this file is **/usr/local/domain/named.ca**; the file name is specified in the boot file, as described above.) The root servers are at the top of the tree, and any part of the tree can be accessed via these root servers. Information about these authorities includes their names and network addresses and a *time_to_live* (data expiration) specification. The *time_to_live* value is normally set high enough to exclude any possibility of the data expiring. This file uses the Standard Resource Record Data format.

The cache initialization file should be identical for all servers whose root authority is the same.

Localhost File

This file specifies the address for the local loopback interface, better known as the *localhost* with the network address **127.0.0.1**. (By convention, this file is named **/usr/local/domain/named.local**; the file name is specified in the boot file, as described above.) This file uses the Standard Resource Record Data format.

Hosts File

This file contains complete host address and resource information about the zone for which a master server is authoritative. Only a primary master server keeps this file on disk: other authoritative servers for the zone cache this information. (By convention, this file is named **/usr/local/domain/named.hosts**; the file name is specified in the boot file for a primary master, as described above.) This file uses the Standard Resource Record Data format.

IN-ADDR.ARPA Database File

This file contains information about the IN-ADDR . ARPA domain, which is a special domain for allowing address to name mapping. (By convention, this file is named **/usr/local/domain/hosts.rev**; the file name is specified in the boot file for a server that is primary for an IN-ADDR . ARPA domain.)

(Refer to the discussion about this domain above and to the complete example file under EXAMPLES below.) This file uses the Standard Resource Record Data format.

Serverless Node Configuration File

The only configuration file used by a serverless node is **/usr/local/domain/resolv.conf**. The file specifies the name servers on the network that service queries for this node. The format of the file is a line specifying the domain, as shown below:

```
domain MySite.COM
```

and a line with the keyword **nameserver** and that server's address for each remote server that handles queries, as shown below:

```
nameserver 3.0.0.11
nameserver 3.0.0.18
```

This file is read when *gethostbyname* or *gethostbyaddr* [see *gethostbyname* (3)] is called.

There should not be a **resolv.conf** file on a host running a name server.

Standard Resource Record Data Format

The records in the name server data files are called resource records. The Standard Resource Record Data format (RR) is specified in RFC882 and RFC973. The following is a general description of these records:

{name} {ttl} addr-class Record_Type Record_Specific_data

Resource records have a standard format shown above. The first field is always the name of the domain record. For some RRs the name may be left blank; in that case it takes on the name of the previous RR. The second field is an optional time to live field. This specifies how long this data is stored in the database. By leaving this field blank the default time to live is specified in the *Start Of Authority* resource record (see below). The third field is the address class; use *IN* for Internet addresses. The fourth field states the type of the resource record. The fields after that are dependent on the type of the RR. Case is preserved in names and data fields when loaded into the name server. All comparisons and lookups in the name server data base are case insensitive.

The following characters have special meanings:

. A free standing dot in the name field refers to the current domain. The origin is appended to a free-standing dot (see below).

@ A free standing @ in the name field denotes the current origin.

\X Where *X* is any character other than a digit (0 through 9), quotes that character so that its special meaning does not apply. For example, "\1." can be used to place a dot character in a label.

\DDD Where each *D* is a digit, is the octet corresponding to the decimal number described by *DDD*. The resulting octet is assumed to be text and is not checked for special meaning.

( ) Parentheses are used to group data that crosses a line. In effect, line terminations are not recognized within parentheses. Note that the left and right parentheses *must each* be preceded by a space, as shown in the examples.

; Semicolon starts a comment; the remainder of the line is ignored.

* An asterisk signifies wildcarding.

Most resource records have the current origin appended to names if they are not terminated by a ".". This is useful for appending the current domain name to the data, such as machine names, but may cause problems where you do not want this to happen. A good rule of thumb follows: if the name is not in of the domain for which you are creating the data file, end the name with a ".".

$INCLUDE

An include line begins with **$INCLUDE**, starting in column 1, and is followed by a file name. This feature is particularly useful for separating different types of data into multiple files. For example,

```
$INCLUDE /usr/named/data/mailboxes
```

The line would be interpreted as a request to load the file **/usr/named/data/mailboxes**. The **$INCLUDE** command does not cause data to be loaded into a different zone or tree. This is simply a way to allow data for a given zone to be organized in separate files. For example, mailbox data might be kept separately from host data using this mechanism.

$ORIGIN

The origin is a way of changing the origin in a data file. The line starts in column 1, and is followed by a domain origin. This is useful for putting more then one domain in a data file. If origin is not specified in a Resource Record Data file, the origin is taken from the boot file.

SOA - Start Of Authority

name	*{ttl}*	*addr-class*	*SOA*	*Origin*	*Person in charge*
@		IN	SOA	hub.MySite.COM.	jack.hub.MySite.COM. (
			1.1	; Serial	
			3600	; Refresh every hour	
			300	; Retry	
			3600000	; Expire	
			3600)	; Minimum	

The *Start of Authority, SOA,* record designates the start of a zone. The *name* is the name of the zone; *origin* is the name of the host on which this data file resides; *person in charge* is the mailing address for the person responsible for the name server [note that a dot (.) rather than an at sign (@) is used in the mailing address). The serial number is the version number of this data file; this number should be incremented whenever a change is made to the data. The name server cannot handle numbers over 9999 after the decimal point. The refresh indicates how often, in seconds, a secondary name server is to check with the primary name server to see if an update is needed. The retry indicates how long, in seconds, a secondary server is to retry after a failure to check for a refresh. Expire is the upper limit, in seconds, that a secondary name server is to use the data before it expires for lack of getting a refresh. Minimum is the default number of seconds to be used for the time to live field on resource records. There should only be one SOA record per zone.

NS - Name Server

{name}	*{ttl}*	*addr-class*	*NS*	*Name servers name*
Sales		IN	NS	jack.Sales.MySite.COM.

The *Name Server* record, NS, lists a name server responsible for a given domain: the name server specified is generally a server responsible for a subdomain (this occurs, for example, in the hosts file) or the root (for example, in the cache file). The first name field lists the domain that is serviced by the listed name server. There should be one NS record for each primary and each secondary master server for the domain. Precedence should be given to a more reliable server: that is, of several servers for a given domain, list them in order of reliability.

A - Address

{name}	*{ttl}*	*addr-class*	*A*	*address*
hub.MySite		**IN**	**A**	**3.0.0.18**
		IN	**A**	**128.1.0.12**

The *Address* record, **A**, lists the address for a given machine; the *name* field specifies the machine name; and the *address* is the network address. There should be one **A** record for each address of the machine.

HINFO - Host Information

{name}	*{ttl}*	*addr-class*	*HINFO*	*Hardware*	*OS*
		IN	**HINFO**	**S/MT**	**CTIX**

Host Information resource record, **HINFO**, is for host specific data. This lists the hardware and operating system that are running at the listed host. It should be noted that only a single space separates the hardware info and the operating system info. If you want to include a space in the machine name, you must enclose the name in quotation marks. There should be one **HINFO** record for each host.

WKS - Well Known Services

{name}	*{ttl}*	*addr-class*	*WKS*	*address*	*protocol*	*list of services*
		IN	**WKS**	**3.0.0.11**	**UDP**	**who route tftp**
		IN	**WKS**	**3.0.0.11**	**TCP**	**(echo telnet**
						discard
						daytime
						chargen ftp
						time finger
						smtp domain)

The *Well Known Services* record, **WKS**, describes the well-known services supported by a particular protocol at a specified address. The list of services and port numbers come from the list of services specified in **/etc/services**. There should be only one **WKS** record per protocol per address.

CNAME - Canonical Name

aliases	*{ttl}*	*addr-class*	*CNAME*	*Canonical name*
PR		**IN**	**CNAME**	**Sales**

Canonical Name resource record, **CNAME**, specifies an alias for a canonical name. An alias should be unique and all other resource records should be associated with the canonical name and not with the alias. Do not create an alias and then use it in other resource records.

PTR - Domain Name Pointer

name	*{ttl}*	*addr-*	*PTR*	*real name*	*class*
22		**IN**	**PTR**	**jack.Sales.MySite.COM.**	

A *Domain Name Pointer* record, **PTR**, allows special names to point to some other location in the domain. The above example of a **PTR** record is used in setting up reverse pointers for the special *IN-ADDR.ARPA* domain. This line is from the example *hosts.rev* file. **PTR** names should be unique to the zone.

MB - Mailbox

name	*{ttl}*	*addr-class*	*MB*	*Machine*
ida		**IN**	**MB**	**Bill.YourSite.COM.**

MB is the *Mailbox* record. This lists the machine where a user wants to receive mail. The name field is the users login; the machine field denotes the machine to which mail is to be delivered. Mail Box names should be unique to the zone.

MR - Mail Rename Name

name	*{ttl}*	*addr-class*	*MR*	*corresponding MB*
Postman		**IN**	**MR**	**ira**

Mail Rename, **MR**, can be used to list aliases for a user. The name field lists the alias for the name listed in the fourth field, which should have a corresponding **MB** record.

MINFO - Mailbox Information

name	*{ttl}*	*addr-class*	*MINFO*	*requests*
grp1		**IN**	**MINFO**	**grp1-REQUEST**

Mail Information record, **MINFO**, creates a mail group for a mailing list. This resource record is usually associated with a mail group *Mail Group*, but can be used with a *Mail Box* record. The *name* specifies the name of the mailbox; the *requests* field is where mail such as requests to be added to a mail group should

be sent; the *maintainer* is a mailbox that should receive error messages. This is particularly appropriate for mailing lists when errors in members names should be reported to a person other than the sender.

MG - Mail Group Member

{mail group name}	*{ttl}*	*addr-class*	*MG*	*member name*
		IN	**MG**	**Roy**

Mail Group, MG, lists members of a mail group. An example for setting up a mailing list follows:

```
grp1    IN      MINFO   grp1-Requestkp.MySite.COM.
        IN      MG      Joe.Sales.MySite.COM.
        IN      MG      Harry.Sales.MySite.COM.
        IN      MG      Jim.Marketing.MySite.COM.
        IN      MG      Nancy.Marketing.MySite.COM.
        IN      MG      Bob.pa.Xerox.COM.
```

MX - Mail Exchanger

name	*{ttl}*	*addr-class*	*MX*	*preference value*	*mailer exchanger*
***.x25.COM.**		**IN**	**MX**	**0**	**mail-gateway.COM.**

Mail Exchanger records, MX, are used to specify a machine that knows how to deliver mail to a machine that is not directly connected to the network. In the example above, **mail-gateway.COM** is a mail gateway that knows how to deliver mail to **x25.COM**, which contains hosts the other machines on the network can not access directly. These machines may have a private connection or use a different transport medium. The preference value is the order that a mailer should follow when there is more then one way to deliver mail to a single machine. See RFC974 for more detailed information.

Wildcard names containing the asterisk character (*) may be used for mail routing with MX records. There are likely to be servers on the network that simply state that any mail to a domain is to be routed through a relay.

FILES

/etc/hosts
/usr/local/domain/hosts.rev
/etc/named
/usr/local/domain/named.boot
/usr/local/domain/named.ca
/usr/local/domain/named.hosts

/usr/local/domain/named.local	
/usr/local/domain/named.pid	Process ID for named written to this file when named is started; used by programs that send signals to named.
/etc/rcopts/NETD	Starts *named* at boot time.
/usr/local/domain/resolv.conf	

SEE ALSO

gethostbyname(3), resolver(3), hosts(4), services(4).
CTIX Network Administrator's Guide.

NOTES

The following signals have the specified effect when sent to the server process using the *kill*(1) command.

SIGHUP Causes server to read named.boot and reload database.

SIGINT Dumps current data base and cache to /usr/tmp/named_dump.db

EXAMPLES

Boot file, **/usr/local/domain/named.boot**, for a primary master server:

```
; Boot file for primary name server
; Note that there should be one primary entry for each SOA record
;
;
; type          domain              source file or host
;
domain          MySite.COM
sortlist                            3.0.0.0 128.1.0.0
primary         MySite.COM          /usr/local/domain/named.hosts
cache           .                   /usr/local/domain/named.ca
primary         0.0.3.IN-ADDR.ARPA  /usr/local/domain/hosts.rev
primary         0.0.127.IN-ADDR.ARPA /usr/local/domain/named.local
```

Boot file, **/usr/local/domain/named.boot**, for a secondary master server:

```
; Boot file for secondary name server
; Note that there should be one primary entry for each SOA record
;
;
; type          domain              source file or host
;
domain          MySite.COM
secondary       MySite.COM          3.0.0.18 3.0.0.14
```

```
                                        /usr/local/domain/hosts.bak
cache           .                       /usr/local/domain/named.ca
secondary       0.0.3.IN-ADDR.ARPA      3.0.0.18 3.0.0.14
primary         0.0.127.IN-ADDR.ARPA    /usr/local/domain/named.local
```

Boot file for caching only server:

```
; Boot file for caching only server
;
;
; type          domain                  source file or host
;
domain          MySite.COM
cache           .                       /usr/local/domain/named.ca
primary         0.0.127.IN-ADDR.ARPA    /usr/local/domain/named.local
```

Serverless node configuration file, **/usr/local/domain/resolv.conf**:

```
domain          MySite.COM
nameserver      3.0.0.18
nameserver      3.0.0.14
```

The hosts file, **/usr/local/domain/named.hosts**, for the **MySite.COM** domain:

```
@           IN      SOA     hub.MySite.COM.  jack.hub.MySite.COM. (
                    1.1     ; Serial
                    3600    ; Refresh every hour
                    300     ; Retry
                    3600000 ; Expire
                    3600 )  ; Minimum
pub         IN      A       128.1.0.15
                    IN      HINFO           S/220
public      IN      CNAME   pub
exec        IN      A       128.1.0.16
            IN      HINFO   S/320
localhost   IN      A       127.1
Finance.    IN      NS      gj.Finance.MySite.COM
            IN      NS      pk.Finance.Mysite.COM
gj.Finance  IN      A       3.0.0.57
            IN      HINFO   S/640 CTIX
pk.Finance  IN      A       3.0.0.46
            IN      HINFO   S/220 CTIX
Sales.      IN      NS      jack.Sales.MySite.COM
```

```
                IN      NS      east.Sales.Mysite.COM
jack.Sales      IN      A       3.0.0.22
                IN      HINFO   S/640 CTIX
east.Sales      IN      A       3.0.0.32
                IN      HINFO   S/220 CTIX
Training.       IN      NS      nh.Training.MySite.COM
                IN      NS      tom.Training.Mysite.COM
nh.Training     IN      A       3.0.0.41
                IN      HINFO   S/MT CTIX
tom.Training    IN      A       3.0.0.42
                IN      HINFO   S/320 CTIX
Marketing.      IN      NS      po.Marketing.MySite.COM
                IN      NS      ja.Marketing.Mysite.COM
po.Marketing    IN      A       3.0.0.65
                IN      WKS     3.0.0.65 UDP who route tftp
                IN      WKS     3.0.0.65 TCP ( echo telnet
                                                discard daytime
                                                chargen ftp
                                                time finger smtp
                                                domain )
                IN      HINFO   S/MT CTIX
ja.Marketing    IN      A       3.0.0.66
                IN      HINFO   S/MT CTIX
grp1            IN      MINFO   grp1-REQUEST    pk.YourSite.COM.
                IN      MG      Joe.Sales.MySite.COM.
                IN      MG      Harry.Sales.MySite.COM.
                IN      MG      Jim.Marketing.MySite.COM.
                IN      MG      Nancy.Marketing.MySite.COM.
                IN      MG      Bob.pa.Xerox.COM.
*.x25.COM.      IN      MX      0               mail-gateway.COM.
Postman         IN      MR      ira
```

Cache file, **/usr/local/domain/named.ca**, for all machines on the local internet (that is, not on the DARPA Internet).

```
; Initial cache data for root domain servers
;
.                       99999999    IN      NS      hub.MySite.COM.
                        99999999    IN      NS      jet.MySite.COM.
; Prep the cache (hardwire the addresses)
hub.MySite.COM.         99999999    IN      A       3.0.0.18
```

```
                    99999999    IN    A     128.1.0.12
jet.MySite.COM.     99999999    IN    A     3.0.0.14
```

Cache file, **/usr/local/domain/named.ca**, for machines on the DARPA Internet:

```
; Initial cache data for root domain servers
;
                    99999999    IN    NS    USC-ISIB.ARPA.
                    99999999    IN    NS    BRL-AOS.ARPA.
                    99999999    IN    NS    SRI-NIC.ARPA.
; Prep the cache (hardwire the addresses)
SRI-NIC.ARPA.       99999999    IN    A     10.0.51
USC-ISIB.ARPA.      99999999    IN    A     10.3.0.52
USC-ISIC.ARPA.      99999999    IN    A     10.0.0.52
BRL-AOS.ARPA.       99999999    IN    A     128.20.1.2
BRL-AOS.ARPA.       99999999    IN    A     192.5.22.82
```

Localhost file, **/usr/local/domain/named.local**:

```
@       IN  SOA     jack.Sales.MySite.COM.   jack.hub.MySite.COM. (
            1.1     ; Serial
            3600    ; Refresh every hour
            300     ; Retry
            3600000 ; Expire
            3600 )  ; Minimum
1       IN  PTR     localhost.
```

One-level IN-ADDR.ARPA domain file, **/usr/local/domain/hosts.rev** (if there were subdomains, there would be an NS record in this file):

```
$ORIGIN             3.IN-ADDR.ARPA
@       IN  SOA     hub.MySite.COM.          jack.hub.MySite.COM. (
            1.1     ; Serial
            3600    ; Refresh every hour
            300     ; Retry
            3600000 ; Expire
            3600 )  ; Minimum
14      IN  PTR     hub.MySite.COM.
18      IN  PTR     jet.MySite.COM.
57      IN  PTR     gj.Finance.MySite.COM.
46      IN  PTR     pk.Finance.MySite.COM.
22      IN  PTR     jack.Sales.MySite.COM.
```

```
32    IN  PTR    east.Sales.MySite.COM.
41    IN  PTR    nh.Training.MySite.COM.
42    IN  PTR    tom.Training.MySite.COM.
65    IN  PTR    po.Marketing.MySite.COM.
66    IN  PTR    ja.Marketing.MySite.COM.
47    IN  PTR    gem.Sales.MySite.COM.
51    IN  PTR    zorba.Sales.MySite.COM.
62    IN  PTR    forecast.Sales.MySite.COM.
63    IN  PTR    news.Finance.MySite.COM.
64    IN  PTR    Personnel.Training.MySite.COM.
71    IN  PTR    west.Sales.MySite.COM.
74    IN  PTR    flora.Marketing.MySite.COM.
```

NAME

nawk - pattern scanning and processing language

SYNOPSIS

nawk [**-F** *re*] [*parameter...*] [*'prog'*] [**-f** *progfile*] [*file...*]

DESCRIPTION

The *nawk* command is a new version of *awk* that provides capabilities unavailable in previous versions. This version will become the default version of *awk* in the next major UNIX system release.

The **-F** *re* option defines the input field separator to be the regular expression *re*.

Parameters, in the form x=... y=... can be passed to *nawk*, where x and y are *nawk* built-in variables (see list below).

nawk scans each input *file* for lines that match any of a set of patterns specified in *prog*. The *prog* string must be enclosed in single quotation marks (') to protect it from the shell. For each pattern in *prog* there may be an associated action performed when a line of a *file* matches the pattern. The set of pattern-action statements may appear literally as *prog* or in a file specified with the **-f** *progfile* option.

Input files are read in order; if there are no files, the standard input is read. The file name - means the standard input. Each input 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 normally made up of fields separated by white space. (This default can be changed by using the FS built-in variable or the **-F** *re* option.) The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

A pattern-action statement has the following form:

pattern { *action* }

Either pattern or action can be omitted. If there is no action with a pattern, the matching line is printed. If there is no pattern with an action, the action is performed on every input line.

Patterns are arbitrary Boolean combinations (**!**, **||**, **&&**, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

expression relop expression
expression matchop regular expression

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

The special patterns BEGIN and END can be used to capture control before the first input line has been read and after the last input line has been read respectively.

Regular expressions are as in *egrep* [see *grep*(1)]. In patterns they must be surrounded by slashes. 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 pattern.

A regular expression can be used to separate fields by using the **-F** *re* option or by assigning the expression to the built-in variable FS. The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value, leading blanks are no longer ignored.

Other built-in variables include:

ARGC	command line argument count
ARGV	command line argument array
FILENAME	name of the current input file
FNR	ordinal number of the current record in the current file
FS	input field separator regular expression (default blank)
NF	number of fields in the current record
NR	ordinal number of the current record
OFMT	output format for numbers (default %.6g)
OFS	output field separator (default blank)
ORS	output record separator (default new-line)
RS	input record separator (default new-line)

An action is a sequence of statements. A statement may be one of the following:

if (conditional) statement [else statement]
while (conditional) statement
do statement while (conditional)

```
for ( expression ; conditional ; expression ) statement
for ( var in array ) statement
delete array[subscript]
break
continue
{ [ statement ] ... }
expression          # commonly variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next      # skip remaining patterns on this input line
exit [expr]          # skip remaining input; exit status is expr
return [expr]
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole input 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 or zero. Array subscripts can 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 *>expression* is present, or on a pipe if | *cmd* is present. The arguments are 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)].

nawk has a variety of built-in functions: arithmetic, string, input/output, and general.

The arithmetic functions are: *atan2*, *cos*, *exp*, *int*, *log*, *rand*, *sin*, *sqrt*, and *srand*. *int* truncates its argument to an integer. *rand* returns a random number between 0 and 1. *srand* (expr) sets the seed value for *rand* to *expr* or uses the time of day if *expr* is omitted.

The string functions are:

gsub(for, repl, in)
: behaves like *sub* (see below), except that it replaces successive occurrences of the regular expression (like the *ed* global substitute command).

index(s, t)
: returns the position in string *s* where string *t* first occurs, or 0 if it does not occur at all.

length(*s*) returns the length of its argument taken as a string, or of the whole line if there is no argument.

match(*s*, *re*) returns the position in string *s* where the regular expression *re* occurs, or 0 if it does not occur at all. RSTART is set to the starting position (which is the same as the returned value), and RLENGTH is set to the length of the matched string.

split(*s*, *a*, *fs*) splits the string *s* into array elements *a*[*1*], *a*[*2*], *a*[*n*], and returns *n*. The separation is done with the regular expression *fs* or with the field separator FS if *fs* is not given.

sprintf(*fmt*, *expr*, *expr*, ...) formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

sub(*for*, *repl*, *in*) substitutes the string *repl* in place of the first instance of the regular expression *for* in string *in* and returns the number of substitutions. If *in* is omitted, *nawk* substitutes in the current record ($0).

substr(*s*, *m*, *n*) returns the *n*-character substring of *s* that begins at position *m*.

The input/output and general functions are:

close(*filename*) closes the file or pipe named *filename*.

cmd | *getline* pipes the output of *cmd* into *getline*; each successive call to *getline* returns the next line of output from *cmd*.

getline sets $0 to the next input record from the current input file.

getline <*file* sets $0 to the next record from *file*.

getline var sets variable *var* instead.

getline var <*file* sets *var* from the next record of *file*.

system(*cmd*) executes *cmd* and returns its exit status.

All forms of *getline* return 1 for successful input, 0 for end of file, and -1 for an error.

nawk also provides user-defined functions. Such functions can be defined (in the pattern position of a pattern-action statement) as

```
function name(args,...) { stmts }
func name(args,...) { stmts }
```

Function arguments are passed by value if scalar and by reference if array name. Argument names are local to the function; all other variable names are global. Function calls may be nested and functions may be recursive. The **return** statement can be used to return a value.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

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

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
{ 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 }
```

Simulate *echo*(1):

```
BEGIN {
        for (i = 1; i < ARGC; i++)
        printf "%s", ARGV[i]
        printf "\n"
        exit
        }
```

Print file, filling in page numbers starting at 5:

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

Command line:

```
nawk -f program n=5 input
```

SEE ALSO

grep(1), lex(1), sed(1), printf(3S).
UNIX System V Release 3.2 Programmer's Guide.

BUGS

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number, add 0 to it; to force it to be treated as a string, concatenate the null string ("") to it.

NAME

ncheck - generate path names from i-numbers

SYNOPSIS

/etc/ncheck [**-i** i-numbers] [**-a**] [**-s**] [file-system]

DESCRIPTION

ncheck with no arguments generates a path-name vs. i-number list of all files on a set of default file systems (see */etc/checklist*). Names of directory files are followed by /..

The options are as follows:

-i Limits the report to only those files whose i-numbers follow.

-a Allows printing of the names . and .., which are ordinarily suppressed.

-s Limits the report to special files and files with set-user-ID mode. This option can be used to detect violations of security policy.

File system must be specified by the file system's special file.

The report should be sorted so that it is more useful.

SEE ALSO

bcheck(1M), fsck(1M), sort(1).

DIAGNOSTICS

If the file system structure is not consistent, **??** denotes the parent of a parentless file and a path-name beginning with ... denotes a loop.

NAME

netstat - show network status

SYNOPSIS

netstat [**-Aan**] [**-f** *address_family*] [*system*] [*core*]
netstat [**-himnrs**] [**-f** *address_family*] [*system*] [*core*]
netstat [**-n**] [**-I** *interface*] *interval* [*system*] [*core*]

DESCRIPTION

The *netstat* command symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. Using the third form, with an *interval* specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces.

The options have the following meanings:

-A With the default display, show the address of any protocol control blocks associated with sockets; used for debugging.

-a With the default display, show the state of all sockets; normally sockets used by server processes are not shown.

-h Show the state of the IMP (Interface Message Processor) 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).

-I *interface*
Show information only about this interface; used with an *interval* as described below.

-m Show statistics recorded by the memory management routines (the network manages a private pool of memory buffers).

-n Show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically). This option can be used with any of the display formats.

-s Show per-protocol statistics.

-r Show the routing tables. When **-s** is also present, show routing statistics instead.

-f *address_family*

Limit statistics or address control block reports to those of the specified *address_family*. The following address families are recognized: *inet*, for **AF_INET**, and *unix*, for **AF_UNIX**.

The arguments, *system* and *core* allow substitutes for the defaults **/unix** and **/dev/kmem**.

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form *host.port* or *network.port* if a socket's address specifies a network but no specific host address. 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 numerically, according to the address family. For more information regarding the Internet "dot format," refer to *rhosts*(4N). 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 addresses 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), whether the route is to a gateway (**G**), and whether the route was created dynamically by a redirect (**D**). Direct routes are created for each interface attached to the local host; the gateway field for such entries shows the address of the outgoing interface. The refcnt field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column for the primary interface (the first interface found during autoconfiguration) and a column summarizing information for all interfaces. The primary interface can be replaced with another interface with the **-I** option. 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

hosts(4N), networks(4N), protocols(4N), services(4N).

BUGS

The notion of errors is ill-defined. Collisions mean something else for the IMP.

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

sendmail(1), aliases(4).

DIAGNOSTICS

If the aliases file is incorrectly formulated, *newaliases* will print diagnostics; the data base will not be updated if there are errors.

NAME

newform - change the format of a text file

SYNOPSIS

newform [**-s**] [**-i** tabspec] [-o tabspec] [**-bn**] [**-en**] [**-pn**] [**-an**] [**-f**] [**-c**char] [**-l**n] [files]

DESCRIPTION

The *newform* command reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for **-s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like "**-e**15 **-l**60" will yield results different from "**-l**60 **-e**15". Options are applied to all *files* on the command line.

-s Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

newform -s -i -l -a -e file-name

-i*tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs*(1). In addition, *tabspec* may be --, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input [see *fspec*(4)]. If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** expects no tabs; if any are found, they are treated as **-1**.

-o*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **-i***tabspec*. If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** means that no spaces will be converted to tabs on output.

-b*n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **-l***n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **-b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform -l1 -b7 file-name
```

-e*n* Same as **-b***n* except that characters are truncated from the end of the line.

-p*n* Prefix *n* characters (see **-c***k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

-a*n* Same as **-p***n* except characters are appended to the end of a line.

-f Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **-o** option. If no **-o** option is specified, the line which is printed will contain the default specification of **-8**.

-c*k* Change the prefix/append character to *k*. Default character for *k* is a space.

-l*n* Set the effective line length to *n* characters. If *n* is not entered, **-l** defaults to 72. The default line length without the **-l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **-i** to expand tabs to spaces).

The **-l1** must be used to set the effective line length shorter than any existing line in the file so that the **-b** option is activated.

DIAGNOSTICS

All diagnostics are fatal.

usage: ...

newform was called with a bad option.

not -s format

There was no tab on one line.

can't open file
Self-explanatory.

internal line too long
A line exceeds 512 characters after being expanded in the internal work buffer.

tabspec in error
A tab specification is incorrectly formatted, or specified tab stops are not ascending.

tabspec indirection illegal
A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

EXIT CODES

0 - normal execution
1 - for any error

SEE ALSO

csplit(1), expand(1), tabs(1), fspec(4).

BUGS

newform normally only keeps track of physical characters; however, for the **-i** and **-o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **-i--** or **-o--**).

If the **-f** option is used, and the last **-o** option specified was **-o--**, and was preceded by either a **-o--** or a **-i--**, the tab specification format line will be incorrect.

NAME

newgrp - log in to a new group

SYNOPSIS

newgrp [-] [group]

DESCRIPTION

newgrp changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp*, regardless of whether it terminated successfully or due to an error condition (that is, unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (PS1) other than **$** (default) and has not exported PS1. After an invocation of *newgrp* , successful or not, their PS1 will now be set to the default prompt string **$**. Note that the shell command *export* [see *sh*(1)] is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry. This is a way to exit the effect of an earlier *newgrp* command.

If the first argument to *newgrp* is a -, the environment is changed to what would be expected if the user actually logged in again as a member of the new group.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

csh(1), login(1), sh(1), group(4), passwd(4), environ(5).

BUGS

There is no convenient way to enter a password into **/etc/group**. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

news - print news items

SYNOPSIS

news [**-a**] [**-n**] [**-s**] [items]

DESCRIPTION

news is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *news* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

-a Causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

-n Causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

-s Causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time.

news is invoked in **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

profile(4), environ(5).
S/Series CTIX Administrator's Guide.

NAME

nfsd, biod - NFS daemons

SYNOPSIS

/etc/nfsd [nservers]

/etc/biod [nservers]

DESCRIPTION

nfsd starts the *NFS* server daemons that handle client filesystem requests. *Nservers* is the number of file system request daemons to start. This number should be based on the average number of simultaneous NFS requests expected on this server.

biod, run on an NFS client, starts *nservers* asynchronous block I/O daemons, which do read-ahead and write-behind of blocks from the client's buffer cache.

nfsd and *biod* are started automatically when the system is booted if there is a zero-length file **/etc/rcopts/KNFS**. *Nserve* defaults to 4 in **/etc/init.d/nfs** for both servers and may be increased by creating the files **/etc/rcopts/NNFSD** and/or **/etc/rcopts/NBIOD** containing the desired number. Increasing these numbers imposes additional system overhead and could be counter productive on lightly loaded NFS systems.

FILES

/etc/init.d/nfs
/etc/rcopts/KNFS

SEE ALSO

mountd(1M), exports(4).

NAME

nfsstat - Network File System statistics

SYNOPSIS

nfsstat [**-csnrz**]

DESCRIPTION

nfsstat displays statistical information about the Network File System (NFS) and kernel-level Remote Procedure Call (RPC) subsystems. Statistics on user-level RPC are not reported. It can also be used to reinitialize this information. If no options are given the default is

nfsstat -csnr

That is, print everything and reinitialize nothing.

OPTIONS

-c Display client information. Only the client side NFS and RPC information will be printed. Can be combined with the **-n** and **-r** options to print client NFS or client RPC information only.

-s Display server information. Works like the **-c** option above.

-n Display NFS information. NFS information for both the client and server side will be printed. Can be combined with the **-c** and **-s** options to print client or server NFS information only.

-r Display RPC information. Works like the **-n** option above.

-z Zero (reinitialize) statistics. Can be combined with any of the above options to zero particular sets of statistics after printing them. The user must have write permission on */dev/kmem* for this option to work.

FILES

/unix	system namelist
/dev/kmem	kernel memory

NAME

nice - run a command at low priority

SYNOPSIS

nice [-increment] command [arguments]

DESCRIPTION

nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., **--10**.

The normal range of negative nice values is **-1** to **-20**; however, an additional range, **-21** to **-28**, is enabled if *rtpenable*(1M) has been executed. Note that the additional range of negative nice values is fixed and pre-emptive and should be used with extreme care.

SEE ALSO

nohup(1), rtpenable(1M), nice(2).

DIAGNOSTICS

nice returns the exit status of the subject command.

BUGS

An *increment* larger than 19 is equivalent to 19.

NAME

nl - line numbering filter

SYNOPSIS

nl [**-h** type] [**-b** type] [**-f** type] [**-v** start#] [**-i** incr] [**-p**] [**-l** num] [**-s** sep] [**-w** width] [**-n** format] [**-d** delim] file

DESCRIPTION

The *nl* command reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

The *nl* command views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (for example, no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

-b*type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are:

-h*type* Same as **-b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

a	number all lines
t	number lines with printable text only
n	no line numbering

p*string* number only lines that contain the regular expression specified in *string*.

Default *type* for logical page body is **t** (text lines numbered).

-f*type* Same as **-b***type* except for footer. Default for logical page footer is **n** (no lines numbered).

-v*start#* *Start#* is the initial value used to number logical page lines. Default is **1**.

-i*incr* *Incr* is the increment value used to number logical page lines. Default is **1**.

-p Do not restart numbering at logical page delimiters.

-l*num* *Num* is the number of blank lines to be considered as one. For example, **-l2** results in only the second adjacent blank being numbered (if the appropriate **-ha**, **-ba**, and/or **-fa** option is set). Default is **1**.

-s*sep* *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.

-w*width* *Width* is the number of characters to be used for the line number. Default *width* is **6**.

-n*format* *Format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes suppressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).

-d*xx* The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **-d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The command:

```
nl -v10 -i10 -d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

NAME

nlsadmin - network listener service administration

SYNOPSIS

nlsadmin -x
nlsadmin [options] net_spec

DESCRIPTION

The *nlsadmin* command administers the network listener process(es) on a machine. Each network has a separate instance of the network listener process associated with it; each instance (and thus, each network) is configured separately. The listener process "listens" to the network for service requests, accepts requests when they arrive, and spawns servers in response to those service requests. The network listener process works with any network (more precisely, with any transport provider) that conforms to the transport provider specification.

The listener supports two classes of service: a General Listener Service, serving processes on remote machines, and a Terminal Login Service, for terminals connected directly to a network. The Terminal Login Service provides networked access to this machine in a form suitable for terminals connected directly to the network. However, this direct terminal service requires special associated software, and is available only with some networks. Currently, it is supported on the DARPA Internet networks, and not through the Terminal Login Service.

The following are valid option parameters or arguments to *nlsadmin*:

net_spec The relative path name under **/dev** of the network special device (for example, STREAMS driver) to open for access to a given network. For example, **inet/tcp** refers to the Transmission Control Protocol (tcp) transport provider within the DARPA Internet (inet) protocol family.

service Uniquely identifies the service the listener is managing. It can be expressed symbolically as a name or numerically as a code. If it is specified as a name, the name is looked up in the file **/usr/net/nls/servcodes** and converted into a code.

address The transport address on which the listener awaits requests for service. Since they await disjointed sets of services, the General Listener Service and the Terminal Login Service require separate listener network addresses. Network addresses are interpreted using a syntax that allows for a variety of addressing formats [see *naddr.d*(5)].

Each unique network (for example, *net_spec*) must have a dedicated listener process. When the network is first connected, *nlsadmin* with the **-i** option must be used to initialize the listener's database, the **-l** and **-t** options used to set the listener's addresses, and the **-a** option used to explicitly add supported services. Once the networks on a new machine have been set up, they need not be set up again. If a unique new network (for example, *net_spec*) is connected, these steps should be repeated.

Once the network is set up, *nlsadmin* can be used to query the status of all or particular listener networks and services; to start or kill the listener process per network; to temporarily enable or disable a service per network; or to permanently add or remove a service per network. Changing the list of services provided by the listener produces immediate changes, while changing an address on which the listener listens has no effect until the listener is restarted.

The following combination of options can be used:

nlsadmin Gives a brief usage message.

nlsadmin -x
Reports the status of all of the listener processes installed on this machine.

nlsadmin *net_spec*
Prints the status of the listener process for *net_spec*.

nlsadmin -q *net_spec*
Queries the status of the listener process for the specified network, and reflects the result of that query in its exit code. If a listener process is active, *nlsadmin* exits with a status of 0; if no process is active, the exit code is 1; in case of error, the exit code is greater than 1.

nlsadmin -v *net_spec*
Prints a verbose report on the servers associated with *net_spec*, giving the service code, status, command, and comment for each.

nlsadmin -z *service net_spec*
Prints a report on the server associated with *net_spec* that has *service*, giving the same information as in the **-v** option.

nlsadmin -q -z *service net_spec*
Queries the status of *service* on network *net_spec*, and exits with a status of 0 if that service is enabled, 1 if that service is disabled, and greater than 1 in case of error.

nlsadmin -l *address net_spec*

Changes or sets the *address* on which the General Listener Service listens for network *net_spec*. This is the address generally used by remote processes to access the servers available through this listener (see the **-a** option, below).

A change of address does not take effect until the next time the listener for that network is started. Since all listeners on a common network must be accessible by remote clients at a commonly-agreed upon address, care should be exercised when changing the listener address. For the *Internet* networks, this known address is built from the service **port** in the shared **/etc/services** file.

If *address* is a dash ("-"), *nlsadmin* reports the address that is currently configured, instead of changing it.

nlsadmin -t *address net_spec*

Changes or sets the *address* on which the Terminal Login Service listens for network *net_spec*. A terminal service address should not be defined unless the appropriate remote login software is available. Otherwise, this option is identical to the **-l** option.

nlsadmin -i *net_spec*

Initializes or changes a listener process for the network specified by *net_spec*: that is, it creates and initializes the files required by the listener. Note that the listener should be initialized only once for a given network, and that doing so does not actually invoke the listener for that network. The listener must be initialized before assigning addresses or services.

nlsadmin [**-m**] **-a** *service* [**-c** *command*] [**-p** *modules*] [**-y** *comment*] *net_spec*

Adds a new *service* to the list of services available through the General Listener Service for network *net_spec*. When a service is added, it is automatically enabled (see the **-e** and **-d** options, below).

If the **-m** option is specified, the entry is added but not be enabled: it is marked as an administrative entry.

The *command* is the full pathname of the command with all arguments that are to be invoked in response to the service request. Since it must appear as a single word to the shell, it should be quoted if arguments are given.

If the **-p** option is specified, f2modules is interpreted as a list of STREAMS modules for the listener to push before starting the added service.

The list is comma-separated with no white space. The modules are pushed in the order they are specified. The module list is specific to the network (*net_spec*) supporting the service.

The optional *comment* is a brief (free-form) description of the service for use in various reports. If the comment contains white space, it must be quoted.

nlsadmin -r *service net_spec*

Removes the entry for the *service* from that listener's list of services. This is normally performed only in conjunction with the removal of a service from a machine.

nlsadmin -e *service net_spec*
nlsadmin -d *service net_spec*

Enables or disables (respectively) the service specified by *service* for the specified network *net_spec*. The service must have previously been added to the listener for that network (see the **-a** option, above). Disabling a service causes subsequent service requests for that service to be denied, but the processes from any prior service requests that are still running continue unaffected.

nlsadmin -s *net_spec*
nlsadmin -k *net_spec*

Starts and kills (respectively) the listener process for the indicated network *net_spec*. These operations are normally performed as part of the system startup and shutdown procedures. Before a listener can be started for a particular network, it must first have been initialized, and an address must have been defined for the General Listener Service (see the **-i** and **-l** options, above). When a listener is killed, processes that are still running as a result of prior service requests continue unaffected.

The listener runs under its own ID of *listen*, with group ID *adm*. This ID must be entered in the system password file; the **HOME** directory listed for that ID is concatenated with *net_spec* to determine the location of the listener configuration information for each network.

The *nlsadmin* command can be invoked by any user to generate reports, but all operations that affect a listener's status or configuration are restricted to the super-user.

FILES

/usr/net/nls/*net_spec*
/etc/hosts
/etc/services

SEE ALSO

servcodes(4), hosts(4), services(4), rfmaster(4).
UNIX System V Release 3.2 Network Programmer's Guide.

NAME

nm - print name list of common object file

SYNOPSIS

nm [**-oxhvnefurpVT**] filename ...

DESCRIPTION

The *nm* command displays the symbol table of each common object file, *filename*. *Filename* may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed:

Name The name of the symbol.

Value Its value expressed as an offset or an address depending on its storage class.

Class Its storage class.

Type Its type and derived type. If the symbol is an instance of a structure or of a union then the structure or union tag will be given following the type (for example, struct-tag). If the symbol is an array, then the array dimensions will be given following the type (for example, char[n][m]). Note that the object file must have been compiled with the **-g** option of the *cc*(1) command for this information to appear.

Size Its size in bytes, if available. Note that the object file must have been compiled with the **-g** option of the *cc*(1) command for this information to appear.

Line The source line number at which it is defined, if available. Note that the object file must have been compiled with the **-g** option of the *cc*(1) command for this information to appear.

Section For storage classes static and external, the object file section containing the symbol (for example, text, data or bss).

The output of *nm* may be controlled using the following options:

-o Print the value and size of a symbol in octal instead of decimal.

-x Print the value and size of a symbol in hexadecimal instead of decimal.

-h Do not display the output header data.

-v Sort external symbols by value before they are printed.

-n Sort external symbols by name before they are printed.

-e Print only external and static symbols.

-f Produce full output. Print redundant symbols (.text, .data, .lib, and .bss), normally suppressed.

-u Print undefined symbols only.

-r Prepend the name of the object file or archive to each output line.

-p Produce easily parsable, terse output. Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **S** (user defined segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (non-external), the type letter is in lower case.

-V Print the version of the nm command executing on the standard error output.

-T By default, *nm* prints the entire name of the symbols listed. Since object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The **-T** option causes *nm* to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both **nm name -e -v** and **nm -ve name** print the static and external symbols in *name*, with external symbols sorted by value.

FILES

*TMPDIR/** temporary files

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

BUGS

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the **-v** and **-n** options should be used only in conjunction with the **-e** option.

SEE ALSO

as(1), cc(1), ld(1), tmpnam(3S), a.out(4), ar(4).

DIAGNOSTICS

nm: name: cannot open
if *name* cannot be read.

nm: name: bad magic
if *name* is not a common object file.

nm: name: no symbols
if the symbols have been stripped from *name*.

NAME

nmountall, numountall - mount, unmount Network File System resources

SYNOPSIS

/etc/nmountall [-] " file-system-table " [...]

/etc/numountall [**-k**]

DESCRIPTION

nmountall is a Network File System command used to mount remote resources according to a *file-system-table*. (**/etc/fstab** is the recommended *file-system-table*.) The special file name "-" reads from the standard input.

Numountall causes all mounted remote resources to be unmounted. The **-k** option sends a SIGKILL signal, via *fuser*(1M), to processes that have files open.

These commands may be executed only by the super-user.

The file-system-table format is as follows:

column 1	block special file name of file system
column 2	mount-point directory
column 3	**-r** if to be mounted read-only
column 4	file system type; "NFS" for Network File System
column 5+	ignored

White-space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

SEE ALSO

fuser(1M), mount(1M), signal(2).

DIAGNOSTICS

Error and warning messages come from *mount*(1M).

NAME

nohup - run a command immune to hangups and quits

SYNOPSIS

nohup command [arguments]

DESCRIPTION

The *nohup* command executes *command* with hangups and quits ignored. If output is not re-directed by the user, both standard output and standard error are sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**.

EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the *nohup* applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored [see *sh*(1)]:

```
nohup file &
```

An example of what the contents of *file* could be follows:

```
sort ofile > nfile
```

SEE ALSO

chmod(1), nice(1), sh(1), signal(2).

WARNINGS

In the case of the following command, *nohup* applies only to command1:

```
nohup command1; command2
```

The following command is syntactically incorrect:

```
nohup (command1; command2)
```

NAME

nroff - format text

SYNOPSIS

nroff [options] [files]

DESCRIPTION

nroff formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers.

If no input file is given, or if the argument - is encountered, *nroff* reads from the standard input file. The *options*, which may appear in any order, but must appear before the *files*, are:

-o*list* Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. 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. (See *BUGS* below.)

-n*N* Number first generated page *N*.

-s*N* Stop every *N* pages. *nroff* will halt *after* every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a line-feed or new-line [new-lines do not work in pipelines, for example, with *mm*(1)]. This option does not work if the output of *nroff* is piped through *col*(1). When *nroff* halts between pages, an ASCII BEL is sent to the terminal.

-r*aN* Set register *a* (which must have a one-character name) to *N*.

-i Read standard input after *files* are exhausted.

-q Invoke the simultaneous input-output mode of the **.rd** request.

-z Print only messages generated by **.tm** (terminal message) requests.

-m*name* Prepend to the input *files* the non-compacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.

-c*name* Prepend to the input *files* the compacted macro files **/usr/lib/macros/cmp.[nt].[dt].***name* and **/usr/lib/macros/ucmp.[nt].***name*.

-k*name* Compact the macros used in this invocation of *nroff*, placing the output in files **[dt].***name* in the current directory

-T*name* Prepare output for specified terminal. Known *name*s are **37** for the (default) TELETYPE® 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, **450** for the DASI 450, **lp** for a (generic) ASCII line printer, **382** for the DTC-382, **4000A** for the Trendata 4000A, **832** for the Anderson Jacobson 832, **X** for a (generic) EBCDIC printer, and **2631** for the Hewlett Packard 2631 line printer.

-e Produce equally-spaced words in adjusted lines, using the full resolution of the particular terminal.

-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.

-u*n* Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*

SEE ALSO

col(1), cw(1), eqn(1), greek(1), mm(1), tbl(1), troff(1), mm(5).

BUGS

nroff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *nroff* generates may be off by one day from your idea of what the date is.

When *nroff* is used with the -o*list* option inside a pipeline it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

NAME

nsquery - Remote File Sharing name server query

SYNOPSIS

nsquery [**-h**] [*name*]

DESCRIPTION

nsquery provides information about resources available to the host from both the local RFS domain and from other RFS domains. All resources are reported, regardless of whether the host is authorized to access them. When used with no options, *nsquery* identifies all resources in the domain that have been advertised as sharable. A report on selected resources can be obtained by specifying *name*, where *name* is:

nodename	The report will include only those resources available from *nodename*.
domain.	The report will include only those resources available from *domain*.
domain.nodename	The report will include only those resources available from *domain.nodename*.

When the name does not include the delimiter ".", it will be interpreted as a *nodename* within the local domain. If the name ends with a delimiter ".", it will be interpreted as a domain name.

The information contained in the report on each resource includes its advertised name (domain.resource), the read/write permissions, the server (nodename.domain) that advertised the resource, and a brief textual description.

When **-h** is used, the header is not printed.

A remote domain must be listed in your **rfmaster** file in order to query that domain.

EXIT STATUS

If no entries are found when *nsquery* is executed, the report header is printed.

ERRORS

If your host cannot contact the domain name server, an error message will be sent to standard error.

SEE ALSO

adv(1M), unadv(1M), rfmaster(4).

NAME

od - octal dump

SYNOPSIS

od [**-bcdosxf**] [file] [[+]offset[.][**b**]]

DESCRIPTION

od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, **-o** is default. The meanings of the format options are:

-b Interpret bytes in octal.

-c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, form-feed=\f, new-line=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.

-d Interpret words in unsigned decimal.

-o Interpret words in octal.

-s Interpret 16-bit words in signed decimal.

-x Interpret words in hex.

-f Interpret bytes in hexadecimal with ASCII listing at side.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If . is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by +.

Dumping continues until end-of-file.

SEE ALSO

dump(1), hd(1).

NAME

pack, pcat, unpack - compress and expand files

SYNOPSIS

pack [-] [**-f**] name ...

pcat name ...

unpack name ...

DESCRIPTION

pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name.z* with the same access modes, access and modified dates, and owner as those of *name*. The **-f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the - argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of - in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each .z file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

- the file appears to be already packed;
- the file name has more than 12 characters;
- the file has links;
- the file is a directory;
- the file cannot be opened;

- no disk storage blocks will be saved by packing;
- a file called *name*.z already exists;
- the .z file cannot be created;
- an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

pcat does for packed files what *cat*(1) does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.z use:

```
pcat name.z
```

or just:

```
pcat name
```

To make an unpacked copy, say *nnn*, of a packed file named *name*.z (without destroying *name*.z) use the command:

```
pcat name >nnn
```

pcat returns the number of files it was unable to unpack. Failure may occur if:

- the file name (exclusive of the .z) has more than 12 characters;
- the file cannot be opened;
- the file does not appear to be the output of *pack*.

unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name*.z (or just *name*, if *name* ends in .z). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

- a file with the "unpacked" name already exists;
- if the unpacked file cannot be created.

SEE ALSO

cat(1).

NAME

passmgmt - password files management

SYNOPSIS

passmgmt -a options name
passmgmt -m options name
passmgmt -d name

DESCRIPTION

The *passmgmt* command updates information in the password files. This command works with both **/etc/passwd** and **/etc/shadow**. If there is no shadow password file the changes done by *passmgmt* will go in **/etc/passwd**.

The *passmgmt -a* form of the command adds an entry for user *name* to the login password files. This command does not create any directory for the new user and the new login remains locked (with the string ***LK*** in the *password* field) until the *passwd*(1M) command is executed to set the password.

The *passmgmt -m* form of the command modifies the entry for user *name* in the login password files. The name field in the **/etc/shadow** entry and all the fields (except the password field) in the **/etc/password** entry can be modified by this command. Only fields entered on the command line are modified. If there is no **/etc/shadow** file, all modifications are made in **/etc/passwd**.

The *passmgmt -d* command deletes the entry for user *name* from the login password files. It does not remove any files the user owns on the system; they must be removed manually.

The login name of the user, *name*, must be unique.

The following options are available:

-c *comment* A short description of the login. It is limited to a maximum of 128 characters and defaults to an empty field. If the comment is more than one word, it must be enclosed in single or double quotation marks.

-h *homedir* Home directory of *name*. It is limited to a maximum of 256 characters and defaults to **/u/***name*.

-u *uid* UID of the *name*. This number must range from 0 to the maximum value for the system. It defaults to the next available UID greater than 100. Without the **-o** option, it enforces the uniqueness of a UID.

-o This option allows a UID to be non-unique. It is used only with the **-u** option.

-g *gid* GID of the *name*. This number must range from 0 to the maximum value for the system. The default is 1.

-s *shell* Login shell for *name*. It should be the full pathname of the program to be executed when the user logs in. The maximum length of *shell* is 256 characters. The default is for this field to be empty and to be interpreted as **/bin/sh**.

-l *logname* This option changes the *name* to *logname* for the **-m** option only.

The total size of each login entry, whether existing or new, is limited to a maximum of 511 bytes in the password files.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

passwd(1), passwd(4), shadow(4).

DIAGNOSTICS

The *passmgmt* command exits with one of the following values:

0 Success.

1 Permission denied.

2 Invalid command syntax. Usage message of the *passmgmt* command is displayed.

3 Invalid argument provided to option.

4 UID in use.

5 Inconsistent password files (for example, *name* is in the **/etc/passwd** file and not in the **/etc/shadow** file, or vice versa).

6 Unexpected failure. Password files unchanged.

7 Unexpected failure. Password file(s) missing.

8 Password file(s) busy. Try again later.

9 *name* does not exist (if **-m** or **-d** is specified), already exists (if **-a** is specified), or *logname* already exists (if **-m -l** is specified).

NOTE

Do not use a colon or carriage return; these characters are interpreted as field separators.

NAME

passwd - change login password

SYNOPSIS

passwd [name]

passwd -s [name]

passwd -l [**-f**] [**-x** max] [**-n** min] name

passwd -d [**-f**] [**-x** max] [**-n** min] name

passwd -s [**-a**]

DESCRIPTION

The *passwd* command changes, sets, or lists attributes of a password associated with the login *name*. Ordinary users can change only the password which corresponds to their login *name*; the super-user can additionally set or change passwords and attributes associated with any login *name*.

When used to change a password, *passwd* prompts ordinary users for their old password, if any; it then prompts for the new password twice. When the old password is entered, *passwd* checks to see if the old password has "aged" sufficiently. Password "aging" is the amount of time (usually a number of days) that must elapse between password changes. If aging is insufficient, *passwd* terminates; see *passwd*(4).

Assuming aging is sufficient, a check is made to ensure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical, the cycle of prompting for the new password is repeated (at most, two more times).

Passwords must be constructed to meet the following requirements:

- Each password must have at least six characters. Only the first eight characters are significant.
- Each password must contain at least two alphabetic (upper- and lowercase) characters and at least one numeric or special character.
- Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.
- New passwords must differ from the old by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

One whose effective user ID is zero is called a super-user; see *id*(1), and *su*(1). Super-users can change any password; therefore, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

Any user can use the -s option to show password attributes for the login *name*.

The format of the display is as follows:

name status mm/dd/yy min max

or, if password aging information is not present,

name status

where:

name Is the login ID of the user.

status Is the password status of *name*: PS stands for passworded or locked, LK stands for locked, and NP stands for no password.

mm/dd/yy Is the date the password was last changed for *name*. Note that all password aging dates are determined by using Greenwhich Mean Time and, therefore, may differ by as much as a day in other time zones.

min Is the minimum number of days required between password changes for *name*.

max Is the maximum number of days the password is valid for *name*.

Only a super-user can use the following options:

-l Locks the password entry for *name*.

-d Deletes the password for *name*. The login *name* is not prompted for a password.

-n Sets the minimum field for *name*. The *min* field contains the minimum number of days between password changes for *name*. If *min* is greater than *max*, the user cannot change the password. Always use this option with the **-x** option, unless *max* is set to -1 (aging disabled), in which case *min* need not be set.

-x Sets the maximum field for *name*. The *max* field contains the number of days the password is valid for *name*. The aging for

name is disabled immediately if *max* is set to -1. If *max* is set to 0, the user must change the password at the next login session, and aging is disabled.

-a Shows password attributes for all entries. Use only with the **-s** option; *name* must not be provided.

-f Forces the user to change the password at the next login by expiring the password for *name*.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

id(1M), login(1), passmgmt(1M), pwconv(1M), su(1M), crypt(3C), passwd(4), shadow(4).

DIAGNOSTICS

The *passwd* command exits with one of the following values:

0 Success.

1 Permission denied.

2 Invalid combination of options (incorrect syntax).

3 Unexpected failure. Password file unchanged.

4 Unexpected failure. Password file(s) missing.

5 Password file(s) busy. Try again later.

6 Invalid argument to option.

WARNINGS

If the optional **/etc/shadow** file exists, *passwd* uses that file instead of **/etc/passwd** to obtain password information. Because the two files store password aging information in different ways, the output from the *passwd* options can differ.

NAME

paste - merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...

paste -d list file1 file2 ...

paste -s [**-d** list] file1 file2 ...

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat*(1) which concatenates vertically, that is, one file after the other. In the last form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if - is used in place of a file name.

The meanings of the options are:

-d Without this option, the new-line characters of each but the last file (or last line in case of the **-s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).

list One or more characters immediately following **-d** replace the default *tab* as the line concatenation character. The list is used circularly, that is, when exhausted, it is reused. In parallel merging (that is, no **-s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), **** (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (for example, to get one backslash, use *-d "\\\\"*).

-s Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **-d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.

- May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

To list directory in one column:

```
ls | paste -d" " -
```

To list directory in four columns:

```
ls | paste - - - -
```

To combine pairs of lines into lines:

```
paste -s -d"\t\n" file
```

SEE ALSO

cut(1), grep(1), pr(1).

DIAGNOSTICS

line too long

Output lines are restricted to 511 characters.

too many files

Except for -s option, no more than 12 input files may be specified.

NAME

path - locate executable file for command

SYNOPSIS

path [-options] *command*

DESCRIPTION

The *path* command provides a quick way to discover which executable file is behind a shell command. It searches each directory mentioned in your **PATH** environment variable until it finds an executable file called *command*.

Any options specified are passed to *ls*(1).

WARNING

The shell [*sh*(1)] hashes the location of certain commands. Therefore, path and type (shell built-in) may give different results.

SEE ALSO

ls(1).

NAME

pg - file perusal filter for CRTs

SYNOPSIS

pg [*-number*] [**-p** *string*] [**-cefns**] [*+linenumber*] [*+/pattern/*] [files...]

DESCRIPTION

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a CRT. (The file name - and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo* (4) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

-number

An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

-p *string*

Causes *pg* to use *string* as the prompt. If the prompt string contains a ''%d'', the first occurrence of ''%d'' in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is '':''.

-c Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo* (4) data base.

-e Causes *pg not* to pause at the end of each file.

-f Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (for example, escape sequences for underlining) generate undesirable results. The *-f* option inhibits *pg* from splitting lines.

-n Normally, commands must be terminated by a *<newline>* character. This option causes an automatic end of command as soon as a command letter is entered.

-s Causes *pg* to print all messages and prompts in standout mode (usually inverse video).

+linenumber
Start up at *linenumber*.

+/pattern/
Start up at the first line containing the regular expression pattern.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)*<newline>* or *<blank>*
This causes one page to be displayed. The address is specified in pages.

(+1) **l** With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or **^L** Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed*(1) are available. They must always be terminated by a *<newline>*, even if the *-n* option is specified.

i/*pattern*/

Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i^*pattern*^

i?*pattern*?

Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

*i***n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w** Display another window of text. If *i* is present, set the window size to *i*.

s*filename*

Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a <*newline*>, even if the *-n* option is specified.

h Help by displaying an abbreviated summary of available commands.

q or **Q** Quit *pg*.

!*command*

Command is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a <*newline*>, even if the *-n* option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the

above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXAMPLE

A sample usage of *pg* in reading system news would be:

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK**, **DEL**, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

/usr/lib/terminfo/?/*	terminal information database
/tmp/pg*	temporary file when input is from a pipe

SEE ALSO

ed(1), grep(1), more(1), terminfo(4).

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options terminal settings may not be restored correctly.

NAME

ping - send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

/etc/ping [**-r**] [**-v**] host [packetsize] [count]

DESCRIPTION

ping is a troubleshooting tool for tracking a single-point hardware or software failure in the Internet. It uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams (''pings'') have an IP and ICMP header, followed by a **struct timeval**, and then an arbitrary number of ''pad'' bytes used to fill out the packet. Default datagram length is 64 bytes, but this may be changed using the command-line option. Other options are:

-r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it [for example, after the interface was dropped by *routed*(1M)].

-v Verbose output. ICMP packets other than ECHO RESPONSE that are received are listed.

When using *ping* for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be ''pinged''. *ping* sends one datagram per second, and prints one line of output for every ECHO_RESPONSE returned. No output is produced if there is no response. If an optional *count* is given, only that number of requests is sent. Round-trip times and packet loss statistics are computed. When all responses have been received or the program times out (with a *count* specified), or if the program is terminated with a SIGINT, a brief summary is displayed.

This program is intended for use in network testing, measurement and management. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is unwise to use *ping* during normal operations or from automated scripts.

SEE ALSO

netstat(1), ifconfig(1M).

NAME

portmap - DARPA port to RPC program number mapper

SYNOPSIS

/etc/portmap

DESCRIPTION

portmap is a server that converts RPC program numbers into DARPA protocol port numbers. It must be running in order to make RPC calls.

When an RPC server is started, it will tell *portmap* what port number it is listening to, and what RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it will first contact *portmap* on the server machine to determine the port number where RPC packets should be sent.

portmap is started automatically when the system is booted if there is a zero-length file **/etc/rcopts/URPC**.

FILES

/etc/rc[23].d/S95nfs
/etc/rcopts/URPC

SEE ALSO

rpcinfo(1M).

BUGS

If *portmap* crashes, all RPC servers must be restarted.

NAME

pr - print files

SYNOPSIS

pr [[-column] [**-w**width] [**-a**]] [**-e**ck] [**-i**ck] [**-drtfp**] [+page] [**-n**ck] [**-o**offset] [**-l**length] [**-s**separator] [**-h**header] [file ...]

pr [[**-m**] [**-w**width]] [**-e**ck] [**-i**ck] [**-drtfp**] [+page] [**-n**ck] [**-o**offset] [**-l**length] [**-s**separator] [**-h**header] file1 file2 ...

DESCRIPTION

pr is used to format and print the contents of a file. If *file* is -, or if no files are specified, *pr* assumes standard input. *pr* prints the named files on standard output.

By default, the listing is separated into pages, each headed by the page number, the date and time that the file was last modified, and the name of the file. Page length is 66 lines which includes 10 lines of header and trailer output. The header is composed of 2 blank lines, 1 line of text (can be altered with **-h**), and 2 blank lines; the trailer is 5 blank lines. For single column output, line width may not be set and is unlimited. For multicolumn output, line width may be set and the default is 72 columns. Diagnostic reports (failed options) are reported at the end of standard output associated with a terminal, rather than interspersed in the output. Pages are separated by series of line feeds rather than form feed characters.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **-s** option is used, lines are not truncated and columns are separated by the *separator* character.

Either *-column* or **-m** should be used to produce multi-column output. **-a** should only be used with *-column* and not **-m**.

Command line options are:

+page Begin printing with page numbered *page* (default is 1).

-column Print *column* columns of output (default is 1). Output appears as if **-e** and **-i** are turned on for multi-column output. May not use with **-m**.

-a Print multi-column output across the page one line per column. *columns* must be greater than one. If a line is too long to fit in a column, it is truncated.

-m Merge and print all files simultaneously, one per column. The maximum number of files that may be specified is eight. If a line is too long to fit in a column, it is truncated. May not use with *-column*.

-d Double-space the output. Blank lines that result from double-spacing are dropped when they occur at the top of a page.

-e*ck* Expand input tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).

-i*ck* In output, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

-n*ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of single column output or each line of **-m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

-w*width* Set the width of a line to *width* character positions (default is 72). This is effective only for multi-column output (*-column* and **-m**). There is no line limit for single column output.

-o*offset* Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

-l*length* Set the length of a page to *length* lines (default is 66). **-l**0 is reset to **-l**66. When the value of *length* is 10 or less, **-t** appears to be in effect since headers and trailers are suppressed. By default, output contains 5 lines of header and 5 lines of trailer leaving 56 lines for user-supplied text. When **-l***length* is used and *length* exceeds 10, then *length*-10 lines are left per page for user supplied text. When *length* is 10 or less, header and trailer output is omitted to make room for user supplied text.

-h *header* Use *header* as the text line of the header to be printed instead of the file name. **-h** is ignored when **-t** is specified or **-l***length* is specified

and the value of *length* is 10 or less. (**-h** is the only *pr* option requiring space between the option and argument.)

-p Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

-f Use single form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

-r Print no diagnostic reports on files that will not open.

-t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page. Use of **-t** overrides the **-h** option.

-s*separator* Separate columns by the single character *separator* instead of by the appropriate number of spaces (default for *separator* is a tab). Prevents truncation of lines on multicolumn output unless **-w** is specified.

EXAMPLES

Print *file1* and *file2* as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

Copy *file1* to *file2*, expanding tabs to columns 10, 19, 28, 37, . . . :

```
pr -e9 -t <file1 >file2
```

Print *file1* and *file2* simultaneously in a two-column listing with no header or trailer where both columns have line numbers:

```
pr -t -n file1 | pr -t -m -n file2 -
```

FILES

/dev/tty* If standard output is directed to one of the special files **/dev/tty***, then other output directed to this terminal is delayed until standard output is completed. This prevents error messages from being interspersed throughout the output.

SEE ALSO

cat(1), pg(1).

NAME

prof - display profile data

SYNOPSIS

prof [**-tcan**] [**-ox**] [**-g**] [**-z**] [**-h**] [**-s**] [**-m** mdata] [prog]

DESCRIPTION

The *prof* command interprets a profile file produced by the *monitor*(3C) function. The symbol table in the object file *prog* (**a.out** by default) is read and correlated with a profile file (**mon.out** by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options **t, c, a,** and **n** determine the type of sorting of the output lines:

-t Sort by decreasing percentage of total time (default).

-c Sort by decreasing number of calls.

-a Sort by increasing symbol address.

-n Sort lexically by symbol name.

The mutually exclusive options **o** and **x** specify the printing of the address of each symbol monitored:

-o Print each symbol address (in octal) along with the symbol name.

-x Print each symbol address (in hexadecimal) along with the symbol name.

The following options may be used in any combination:

-g Include non-global symbols (static functions).

-z Include all symbols in the profile range [see *monitor*(3C)], even if associated with zero number of calls and zero time.

-h Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)

-s Print a summary of several of the monitoring parameters and statistics on the standard error output.

-m mdata

Use file *mdata* instead of **mon.out** as the input profile file.

A program creates a profile file if it has been loaded with the **-p** option of *cc*(1). This option to the *cc* command arranges for calls to *monitor*(3C) at the beginning and end of execution. It is the call to *monitor* at the end of execution that causes a profile file to be written. The number of calls to a function is tallied if the **-p** option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable PROFDIR. If PROFDIR does not exist, "mon.out" is produced in the directory that is current when the program terminates. If PROFDIR = string, "string/pid.progname" is produced, where *progname* consists of argv[0] with any path prefix removed, and *pid* is the program's process id. If PROFDIR is the null string, no profiling output is produced.

A single function may be split into subfunctions for profiling by means of the MARK macro [see *prof*(5)].

FILES

mon.out	for profile
a.out	for namelist

SEE ALSO

cc(1), exit(2), profil(2), monitor(3C), prof(5).

WARNING

The times reported in successive identical runs may show variances of 20% or more, because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may "beat" with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely.

The times for static functions are attributed to the preceding external text symbol if the **-g** option is not used. However, the call counts for the preceding function are still correct, that is, the static function call counts are not added in with the call counts of the external function.

CAVEATS

Only programs that call *exit*(2) or return from *main* will cause a profile file to be produced, unless a final call to monitor is explicitly coded.

The use of the **-p** option to *cc*(1) to invoke profiling imposes a limit of 600 functions that may have call counters established during program execution. For more counters you must call *monitor*(3C) directly. If this limit is exceeded, other data will be overwritten and the **mon.out** file will be corrupted. The number of call counters used will be reported automatically by the *prof* command whenever the number exceeds 5/6 of the maximum.

NAME

profiler: prfld, prfstat, prfdc, prfsnap, prfpr - operating system profiler

SYNOPSIS

/etc/prfld [namelist]

/etc/prfstat on

/etc/prfstat off

/etc/prfdc file [period [off_hour]]

/etc/prfsnap file

/etc/prfpr file [cutoff [namelist]]

S/640 Only:
/etc/prfstat time

DESCRIPTION

The *prfld*, *prfstat*, *prfdc*, *prfsnap*, and *prfpr* programs form a system of programs to facilitate an activity study of the CTIX operating system. A kernel configured with kernel profiling must be used: the **pfr** driver may be loaded with *lddrv*(1M).

The *prfld* program is used to initialize the recording mechanism in the system. It generates a table containing the starting address of each system subroutine as extracted from *namelist*.

The *prfstat* program is used to enable or disable the sampling mechanism. Profiler overhead is less than one percent as calculated for 500 text addresses. Note that *prfstat* also reveals the number of text addresses being measured.

Addresses are sampled every clock tick (definition for *Hz* is given in *param.h*). S/640 systems allow sampling every *time* microsecond: the lower limit is 100 microsecond intervals.

The *prfdc* and *prfsnap* programs perform the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed. The *prfdc* program stores the counters into *file* every *period* minutes and turns off at *off_hour* (valid values for *off_hour* are **0-24**). The *prfsnap* program collects data at the time of invocation only, appending the counter values to *file*.

The *prfpr* program formats the data collected by *prfdc* or *prfsnap*. Each text address is converted to the nearest text symbol (as found in *namelist*) and is printed if the percent activity for that range is greater than *cutoff*.

FILES

/dev/prf	interface to profile data and text addresses
/etc/lddrv/unix.exec	default for namelist file

SEE ALSO

prf(7).

NAME

prs - print an SCCS file

SYNOPSIS

prs [**-d**[dataspec]] [**-r**[SID]] [**-e**] [**-l**] [**-c**[date-time]] [**-a**] files

DESCRIPTION

The *prs* command prints, on the standard output, parts or all of an SCCS file [see *sccsfile*(4)] in a user-supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of **-** is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to *prs*, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

-d*[dataspec]* Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.

-r*[SID]* Used to specify the *SCCS* ID*entification* (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

-e Requests information for all deltas created *earlier* than and including the delta designated via the **-r** keyletter or the date given by the **-c** option.

-l Requests information for all deltas created *later* than and including the delta designated via the **-r** keyletter or the date given by the **-c** option. The cutoff date-time **-c**[cutoff]] is in the form:

YY[MM[DD[HH[MM[SS]]]]]

-c*[date-time]* Units omitted from the date-time default to their maximum possible values; that is, **-c7502** is equivalent to **-c750228235959**. Any number of non-numeric characters can separate the various two-digit pieces of the *cutoff* date:

-c77/2/2 9:22:25

-a Requests printing of information for both removed, that is, delta type = *R*, [see *rmdel*(1)] and existing, that is, delta type = *D*, deltas. If the **-a** keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file [see *sccsfile*(4)] have an associated data keyword. There is no limit on the number of times a data keyword can appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords.

A tab is specified by **\t** and carriage return/new-line is specified by **\n**. The default data keywords are:

":Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S

:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS: ...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS: ...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS: ...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes* or *no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes* or *no*	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	*yes* or *no*	S
:J:	Joint edit flag	"	*yes* or *no*	S
:LK:	Locked releases	"	:R: ...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes* or *no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

The command

prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file

can produce the following on standard output:

```
Users and/or user IDs for
s.file
are:
xyz
131
abc
```

The command

prs -d"Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file

can produce the following on standard output:

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

As a *special case*, the command:

prs s.file

can produce the following on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the **-a** keyletter.

FILES

/tmp/pr*?????*

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(4).
UNIX System V Release 3.2 Programmer's Guide.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps - report process status

SYNOPSIS

ps [options]

DESCRIPTION

ps prints certain information about active processes. Without *options*, information is printed about processes associated with the controlling terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

Options accept names or lists as arguments. Arguments can be either separated from one another by commas or enclosed in double quotes and separated from one another by commas or spaces. Values for *proclist* and *grplist* must be numeric.

The *options* are given in descending order according to volume and range of information provided:

-e Print information about every process now running.

-d Print information about all processes except process group leaders.

-a Print information about all processes most frequently requested: all those except process group leaders and processes not associated with a terminal.

-f Generate a full listing. (Normally, a short listing containing only process ID, terminal (''tty'') identifier, cumulative execution time, and the command name is printed.) See below for significance of columns in a full listing.

-l Generate a long listing. (See below.)

-n *namelist* Take argument signifying an alternate system *namelist* file in place of **/unix**.

-t *termlist* List only process data associated with the terminal given in *termlist*. Terminal identifiers may be specified in one of two forms: the device's file name (for example, **tty004**) or, if the device's file name starts with **tty**, just the digit identifier (for example, **004**).

-p *proclist* List only process data whose process ID numbers are given in *proclist*.

-u *uidlist* List only process data whose user ID number or login name is given in *uidlist*. If the **-l** option was used, the numerical UID is printed. If the **-f** option was used, the login name is printed.

-g *grplist* List only process data whose process group leader's ID number(s) appears in *grplist*. (A group leader is a process whose process ID number is identical to its process group ID number. A login shell is a common example of a process group leader.)

Under the **-f** option, *ps* tries to determine the command name and arguments given when the process was created by examining the user block. Failing this, the command name is printed, as it would have appeared without the **-f** option, in square brackets.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (full or long, respectively) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do not determine which processes will be listed.

F (l) Flags (hexadecimal and additive) associated with the process:

- 00 Process has terminated: process table entry now available.
- 01 A system process: always in primary memory.
- 02 Parent is tracing process.
- 04 Tracing parent's signal has stopped process: parent is waiting [*ptrace*(2)].
- 10 Process is currently in primary memory.
- 20 Process currently in primary memory: locked until an event completes.
- 2000 Process being swapped.

S (l) The state of the process:

- O Process is running on a processor.
- S Sleeping: process is waiting for an event to complete.
- R Runnable: process is on run queue.

I Idle: process is being created.

Z Zombie state: process terminated and parent not waiting.

T Traced: process stopped by a signal because parent is tracing it.

X SXBRK state: process is waiting for more primary memory.

UID (f,l) The user ID number of the process owner (the login name is printed under the **-f** option).

PID (all) The process ID of the process (this datum is necessary in order to kill a process).

PPID (f,l) The process ID of the parent process.

C (f,l) Processor utilization for scheduling.

PRI (l) The priority of the process (higher numbers mean lower priority).

NI (l) Nice value, used in priority computation.

ADDR (l) The physical page number of the process's user page.

SZ (l) The size (in 4-Kbyte pages) of the swappable process's image in main memory.

WCHAN (l) The address of an event for which the process is sleeping, or in SXBRK state, (if blank, the process is running).

STIME (f) The starting time of the process, given in hours, minutes, and seconds. (A process begun more than twenty-four hours before the *ps* inquiry is executed is given in months and days.)

TTY (all) The controlling terminal for the process (the message, **?**, is printed when there is no controlling terminal). The **s** prefix implies a shell layer; the **p** prefix implies a virtual terminal; and the **w** prefix implies a CTAM window (**/dev/wxt/***), the **r** prefix indicates a terminal controlled by an RIOP.

TIME (all) The cumulative execution time for the process.

COMMAND (all) The command name (the full command name and its arguments are printed under the **-f** option).

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

FILES

/dev	
/dev/sxt/*	
/dev/wxt/*	
/dev/tty*	
/dev/rtty*	
/dev/kmem	kernel virtual memory
/dev/swap	the default swap device
/dev/mem	memory
/etc/passwd	UID information supplier
/etc/ps_data	internal data structure
/unix	system namelist

SEE ALSO

acctcom(1), getty(1M), kill(1), nice(1).

WARNING

Things can change while *ps* is running; the snap-shot it gives is only true for a split-second, and it may not be accurate by the time you see it. Some data printed for defunct processes is irrelevant.

If no *termlist*, *proclist*, *uidlist*, or *grplist* is specified, *ps* checks *stdin*, *stdout*, and *stderr* in that order, looking for the controlling terminal and attempts to report on processes associated with the controlling terminal. In this situation, if *stdin*, *stdout*, and *stderr* are all redirected, *ps* does not find a controlling terminal, so there is no report.

On a heavily loaded system, *ps* may report an *lseek(2)* error and exit. *ps* may seek to an invalid user area address: having obtained the address of a process' user area, *ps* may not be able to seek to that address before the process exits and the address becomes invalid.

ps -ef may not report the actual start of a tty login session, but rather an earlier time, when a getty was last respawned on the tty line.

If the user specifies the **-n** flag, the real and effective UID/GID is set to the real UID/GID of the user invoking *ps*.

NAME

ptx - permuted index

SYNOPSIS

ptx [options] [input [output]]

DESCRIPTION

The *ptx* command generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). 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 each line. The *ptx* output is in the following form:

.xx "tail" "before keyword" "keyword and after" "head"

where **.xx** is assumed to be an *nroff* or *troff*(1) macro provided by the user, or provided by the *mptx*(5) macro package. The *before keyword* and *keyword and after* fields incorporate as much of the line as fits around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The index for this manual was generated using *ptx*.

The following *options* can be applied:

-f Fold upper and lower case letters for sorting.

-t Prepare the output for the phototypesetter.

-w *n* Use the next argument, *n*, as the length of the output line. The default line length is 72 characters for *nroff* and 100 for *troff*.

-g *n* Use the next argument, *n*, as the number of characters that *ptx* reserves in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3.

-o *only* Use as keywords only the words given in the *only* file.

-i *ignore* Do not use as keywords any words given in the *ignore* file. If the **-i** and **-o** options are missing, use **/usr/lib/eign** as the *ignore* file.

-b *break* Use the characters in the *break* file to separate words. Tab, newline, and space characters are *always* used as break characters.

-r Take any leading non-blank 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.

FILES

/bin/sort
/usr/lib/eign
/usr/lib/tmac/tmac.ptx

SEE ALSO

nroff(1), troff(1), mm(5), mptx(5).

BUGS

Line length counts do not account for overstriking or proportional spacing.
Lines that contain tildes (~) are botched, because *ptx* uses that character internally.

NAME

pwck, grpck - password/group file checkers

SYNOPSIS

/etc/pwck [file]

/etc/grpck [file]

DESCRIPTION

The *pwck* command scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and the program-to-use-as-Shell exist. The default password file is **/etc/passwd**.

Grpck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group**.

FILES

/etc/group
/etc/passwd

SEE ALSO

group(4), passwd(4).
S/Series CTIX Administrator's Guide.

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

NAME

pwconv - install and update /etc/shadow with information from /etc/passwd

SYNOPSIS

pwconv

DESCRIPTION

The *pwconv* command creates and updates **/etc/shadow** with information from **/etc/passwd**. If the **/etc/shadow** file does not exist, *pwconv* creates it with information from **/etc/passwd**. The command populates **/etc/shadow** with the user's login name, password, and password aging information. If password aging information does not exist in **/etc/passwd** for a given user, none is added to **/etc/shadow**; however, the "last changed" information is always be updated.

If the **/etc/shadow** file does exist, the following tasks are performed:

- Entries in the **/etc/passwd** file but not in the **/etc/shadow** file are added to the **/etc/shadow** file.
- Entries in the **/etc/shadow** file but not in the **/etc/passwd** file are removed from **/etc/shadow**.
- Password attributes (such as password and aging information) that exist in an **/etc/passwd** entry are moved to the corresponding entry in **/etc/shadow**.

The *pwconv* program is a privileged system command that can be executed only by the super-user. The *passwd* command should be used to add or change password aging information or passwords.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

passwd(1M), passmgmt(1M), pwunconv(1M).

DIAGNOSTICS

The *pwconv* command exits with one of the following values:

0	Success.
1	Permission denied.
2	Invalid command syntax.
3	Unexpected failure. Conversion not done.
4	Unexpected failure. Password file(s) missing.
5	Password file(s) busy. Try again later.

NAME

pwd - working directory name

SYNOPSIS

pwd

DESCRIPTION

pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1).

DIAGNOSTICS

"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred your system administrator.

NAME

pwunconv - install and update /etc/shadow with information from /etc/passwd

SYNOPSIS

pwunconv

DESCRIPTION

The *pwunconv* command converts a system from a two-password file scheme (**/etc/passwd** and **/etc/shadow**) to a one-password file scheme (**/etc/passwd**). It updates **/etc/passwd** with password information from **/etc/shadow**. If aging information is present in **/etc/shadow**, the password aging information in **/etc/passwd** is also updated.

The total size of a login entry for the password file is limited to a maximum of 511 bytes.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

passwd(1M), passmgmt(1M), pwconv(1M).

DIAGNOSTICS

The *pwunconv* command exits with one of the following values:

0	Success.
1	Permission denied.
2	Invalid command syntax.
3	Unexpected failure. Conversion not done.
4	Unexpected failure. Password file(s) missing.
5	Password file(s) busy. Try again later.

NAME

qinstall - install and verify software using the mkfs(1) proto file database

SYNOPSIS

/usr/local/bin/qinstall -c [**-esoimk**] proto root

/usr/local/bin/qinstall -g [**-p** prefix] [**-t** num] root

/usr/local/bin/qinstall -b [**-a**] [**-t** num] proto1 proto2

/usr/local/bin/qinstall -u [**-oeiw**] proto1 proto2

/usr/local/bin/qinstall -r [**-kq**] proto root from_rfile to_rfile

DESCRIPTION

qinstall is used to package software on distribution media, to install software, and to verify the correctness of the installation. Output from *qinstall* goes to standard output. *Root* must be a full pathname or ".".

The following options are recognized by *qinstall*:

-V output additional, verbose messages to *stderr*.

-c check whether files under *root* match files in *proto* for owner and permission. This option is used primarily to verify the correctness of an installation, but it is also used in the software distribution packaging process.

- **-k** allow changing owners/modes of CORE directories.
- **-o** print omissions from *root/proto2*.
- **-s** set permissions and owners to be correct if incorrect.
- **-e** print extra files not found in *proto/proto1*.
- **-i** ignore differences in special files.
- **-m** check mounted file systems as well.

-g generate a proto file from *root*. This option is used in the software distribution packaging process.

- **-p** *prefix* add *prefix* to path names.
- **-t** *num* specify number of tabs to indent.

-b blend the two proto files together into one.

- **-a** add a group name generated from the name of proto2 (converted to upper case).

-t *num* specify number of tabs to indent.

-u checks if files in proto1 match files in proto 2 for owner and permissions.

-o print omissions from *root/proto2*.

-e print extra files not found in *proto/proto1*.

-i ignore differences in special files.

-w check groups as well.

-r replace file *to_rfile* in *root* with contents of *from_rfile*, keeping permissions as in *proto*. *To_rfile* path must be a full path name as in the proto file, and will be offset with *root*. Multiple from/to pairs may be specified. This option is used to install customizable software.

-k allow changing owners/modes of CORE directories.

-q query before replace. Options are to replace *to_rfile* with *from_rfile*, to save *from_rfile*, to ignore *to_rfile*, to perform an *sdiff*(1) between the two files or to replace *to_rfile* with the previous diff.

EXAMPLE

A sample proto file created with the **-g** option follows.
(qinstall -g . > ../proto)

```
/mkboot
0 0
d--777 2 2
install    d--775 0 0
    lsamRel        ---444 0 0 /install/lsamRel
    $
usr        d--775 2 2
    include        d--775 2 2
        iserc.h    ---444 2 2 /usr/include/iserc.h
        isam.h     ---444 2 2 /usr/include/isam.h
        $
    lib    d--775 2 2
        lsam       d--775 2 2
            lsamConfig     ---755 2 2 /usr/lib/lsam/lsamConfig
            lsamCreate     ---755 2 2 /usr/lib/lsam/lsamCreate
            lsamProtect    ---755 2 2 /usr/lib/lsam/lsamProtect
            lsamReorg      ---755 2 2 /usr/lib/lsam/lsamReorg
            lsamStat       ---755 2 2 /usr/lib/lsam/lsamStat
```

```
            lsamStop        ---755 2 2 /usr/lib/lsam/lsamStop
            lsamTransfer    ---755 2 2 /usr/lib/lsam/lsamTransfer
            lxFilter        ---755 2 2 /usr/lib/lsam/lxFilter
            lxSpec          ---755 2 2 /usr/lib/lsam/lxSpec
            lsam            ---755 2 2 /usr/lib/lsam/lsam
            $
        liblsam.a           ---444   2 2 /usr/lib/liblsam.a
        $
    $
$
```

SEE ALSO

qlist(1), ctinstall(1), mkfs(1).

BUGS

qinstall invoked with the **-m** option on an inconsistent file system produces error messages of the form "filename: cannot stat".

NAME

qlist - print out file lists from proto file; set links based on lines in proto file.

SYNOPSIS

/usr/local/bin/qlist -m [**-d** dir] [**-d** dir] [**-o**] [**-p** prefix] proto

/usr/local/bin/qlist -l dir [**-p** prefix] proto

/usr/local/bin/qlist -n proto root

/usr/local/bin/qlist -s proto root

DESCRIPTION

The *qlist* command is used in the distribution software packaging process and in the software installation process. It makes lists of files from proto files created by *qinstall*(1). Lists are based on the files' group identifiers and types. *qlist* also sets links based on lines in the proto file during software installation.

qlist understands extended proto files, in which a line beginning with **:L** indicates that the first file named is a link to the second file. Other lines beginning with : are comments. The last field on a line in an extended proto file is a group identifier of 9 or fewer characters, such as "WP" for the Word Processor product. The following symbols appearing immediately after the group identifier designate the file's type and have the following meanings:

+ designates a customizable file, such as **/etc/passwd**. This type of file is one which the user may or may not want to install over his existing version. This type of file can be installed with the **-rq** option of *qinstall*(1).

- designates a zero-length file. The specified file should not be used when updating an existing system; rather, it should be used for raw, or first installs only.

@ implies an update but no query from *qinstall*(1). This symbol is used for files required by the installation tools for installation and for possible text busy files.

< designates an optional file, or a file requiring special installation such as a hardware configuration-dependent file. Its associated special installation scripts are GROUP.opt and GROUP.ins, where *GROUP* represents the group name.

< id designates a file of the above category which has special installation scripts named GROUPid.opt, GROUPid.ins, where *GROUP* represents the group name. *Id* can be 5 or fewer characters. The total number of characters in *GROUP and id must* be 10 or fewer.

The following options are recognized by *qlist*:

-V output additional, verbose messages to *stderr*.

-m make file lists from *proto* file. This option is used in packaging software.

-o print files in no group.

-d use *dir* as location for file lists.

-p use prefix when printing (default = **./**).

File lists output with the **-m** option for group "WP" are named as follows:

+	WP.cust
-	WP.noup
@	WP.noqu
<	WP.fopt, WP.flst
< id	WP.fopt, WPid.lst
the rest	WP

-l list files in directory *dir* from *proto* file to *stdout*.

-p use prefix when printing (default = **./**).

-s set links in *root* directory which are indicated by **:L** lines in *proto* file. *Root* must be a rooted path name or ".". This option is used in software installations.

-n mknod special files if they do not exist allready.

EXAMPLE

A sample extended proto file follows. Note that the files **Document** and **Gloss** are really links to the file **Admin**, as indicated by **:L** at the beginning of these lines. Also note that the lines ending in **<** designate optionally installed, or specially installed files.

```
/mkboot
0 0
d--777 2 2
install                 d--775 0 0
    WPRel               ---444 0 0 /install/WPRel       WP
    $
oa                      d--775 0 0
    .Key    d--755 0 0
            Admin       ---444 2 2 /oa/.Key/Admin       CTIXOA
:L          Document            /oa/.Key/Admin          CTIXOA
:L          Gloss               /oa/.Key/Admin          CTIXOA
            $
    .Document           d--775 0 0
            Recruit     ---666 2 2 /oa/.Document/RecruitWP
            $
    .Gloss  d--775 0 0
            Sample      ---666 2 2 /oa/.Gloss/Sample    WP
            $
    Centronix           ---555 2 2 /oa/Centronix        WP<     sys
    ImagenDriver        ---555 2 2 /oa/ImagenDriver     WP<     sys
    SerialDriver        ---555 2 2 /oa/SerialDriver     WP<     sys
    abs_rel---555 2 2 /oa/abs_rel WP<       propt
    ctospool            ---555 2 2 /oa/ctospool         WP
    def_wp---555 2 2 /oa/def_wp WP<         propt
    spoolstat           ---555 2 2 /oa/spoolstat        WP
    wp_def---555 2 2 /oa/wp_def WP<         propt
    wp_edit             ---555 2 2 /oa/wp_edit          WP
    wp_merge            ---555 2 2 /oa/wp_merge         WP
    wp_print            ---555 2 2 /oa/wp_print         WP
    wp_review           ---555 2 2 /oa/wp_review        WP
    wpp_band            ---555 2 2 /oa/wpp_band         WP<     propt
    wpp_canprt          ---555 2 2 /oa/wpp_canprt       WP
    wpp_diablo          ---555 2 2 /oa/wpp_diablo       WP<     propt
    wpp_imagen          ---555 2 2 /oa/wpp_imagen       WP<     propt
    wpp_laser           ---555 2 2 /oa/wpp_laser        WP<     propt
    wpp_necspin         ---555 2 2 /oa/wpp_necspin      WP<     propt
    wpp_prtsh           ---555 2 2 /oa/wpp_prtsh        WP
    $
$
```

SEE ALSO

qinstall(1), ctinstall(1).

NAME

ratfor - rational FORTRAN dialect

SYNOPSIS

ratfor [options] [files]

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

ratfor also provides the following syntactic enhancements 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 to caller from function:

return (expression)

define:

define *name replacement*

include:

include *file*

The **-h** option causes quoted strings to be turned into **27H** constructs. The **-C** option copies comments to the output and attempts to format it neatly. Normally, continuation lines are marked with a **&** in column 1; the option **-6x** makes the continuation character **x** and places it in column 6.

SEE ALSO

efl(1).

Brian W. Kernighan; P. J. Plauger. *Software Tools.* Reading, Mass.: Addison-Wesley, 1976.

NAME

rc0 - run commands performed to stop the operating system

SYNOPSIS

/etc/rc0

DESCRIPTION

This file is executed at each system state change that needs to have the system in an inactive state. It is responsible for those actions that bring the system to a quiescent state, traditionally called "shutdown."

The following five system states require the */etc/rc0* procedure: **0**, **1**, **4**, **5**, and **6**. The procedure is run each time a change to one of these states occurs. The entry in **/etc/inittab** should read as follows:

```
s0:01456:wait:/etc/rc0 >/dev/console 2>&1
```

Some of the actions performed by **/etc/rc0** are carried out by files beginning with **K** in **/etc/rc0.d**. (All **/etc/rc0.d/K*** files are merely links to files in **/etc/init.d**: the **K** at the beginning of the file name designates it as a system "stop" procedure.) These K* files are executed in ASCII order, and each terminates some system service. The combination of commands in **/etc/rc0** and files in **/etc/rc0.d** determines how the system is shut down. (Note that **/etc/rc0** actually executes files beginning with **S** in **/etc/rc0.d** if there are any such files. The CTIX distribution provides only **K** files in **/etc/rc0.d**.)

/etc/rc0 performs the following:

- Announces that the system is coming down.
- Stops error logging, the spooler, and any other system service for which there is an executable file named **/etc/rc0.d/K***.
- Kills all active processes.
- Unmounts file systems.
- Instructs users to wait for a message that it is okay to stop or reset the processor.

Depending on which system state the system ends up in, the entries in **/etc/inittab** direct what happens next. If the **/etc/inittab** has not defined any other actions to be performed, as in the case of system state 0, a shell prompt is displayed. The command can be used only by the super-user.

NOTE

Although **/etc/rc0** is an ASCII text commands file, it is not meant to be "configurable." System services are terminated through procedures in **/etc/rc.d/K*** files; these files check for the presence of files in the **/etc/rcopts**

directory. Therefore, configurability is provided at the level of an "rcopt:" that is, the system administrator creates an rcopt file with a specific name (see the **README** file in that directory), and pre-defined, non-configurable scripts perform all the necessary start and stop procedures.

FILES

/etc/init.d/* procedures to be started and stopped when there is a change in run-level; linked to files in **/etc/rc?.d**; whether the file is executed at a particular run-level is determined by whether there is a K (terminate) or S (start) file in the **/etc/rc?.d** directory that corresponds to the run-level

/etc/rc0.d/K* files executed in ascii order when **/etc/rc0** is called; each script terminates a particular system service

/etc/rcopts/* their presence (and in some cases their contents) checked by **/etc/rc?.d/*** scripts to determine what system service (such as the spooler) must be started or stopped.

SEE ALSO

init(1M), killall(1M), rc2(1M), shutdown(1M). inittab(4).
S/Series CTIX Administrator's Guide.

NAME

rc2, rc3 - run commands performed for multi-user environment

SYNOPSIS

/etc/rc2

/etc/rc3

DESCRIPTION

/etc/rc2 is executed via an entry in **/etc/inittab** and is responsible for those initializations that bring the system to a ready-to-use state, traditionally state 2, called the "multi-user" state. The CTIX distribution includes **/etc/rc3** for use in state 3, the multi-user state for Remote File Sharing: the distribution **/etc/rc2** and **/etc/rc3** files are identical except for references to the state number.

The actions performed by **/etc/rc[23]** are found in files in the directory **/etc/init.d**: these files are linked to files with a **K***nn* prefix or an **S***nn* prefix in **/etc/rc[23].d**. The files are executed by **/bin/sh** in ASCII sort-sequence order, **K** files with the parameter **stop** and then **S** files with the parameter **start**. (See FILES for more information).

The functions done by the **/etc/rc[23]** command and associated **/etc/rc[23].d** files (depending on the presence of particular files in the **/etc/rcopts** directory) include:

- Setting-up and mounting file systems.
- Cleaning up (remaking) the **/tmp** and **/usr/tmp** directories.
- Setting the system node name and Internet hostname.
- Starting network daemons.
- Starting the *cron* daemon by executing **/etc/cron**.
- Cleaning up (deleting) uucp locks status, and temporary files in the **/usr/spool/uucp** directory.

If Remote File Sharing is installed, **K***nn* files in **/etc/rc2.d** stop Remote File Sharing and **S***nn* files in **/etc/rc3.d** start Remote File Sharing.

EXAMPLES

The following are prototypical files found in **/etc/rc[23].d**. These files are prefixed by an S and a number indicating the execution order of the files.

MOUNTFILESYS

```
# Set up and mount file systems

cd /
/etc/mountall /etc/fstab
```

```
RMTMPFILES
                .
                .
                .
        # clean up /tmp
        rm -rf /tmp
        mkdir /tmp
        chmod 777 /tmp
        chgrp sys /tmp
        chown sys /tmp
                .
                .
                .
uucp
        #   clean-up uucp locks, status,
            and temporary files
                .
                .
                .
        rm -rf /usr/spool/locks/*
```

The file **/etc/TIMEZONE** is included early in */etc/rc[23]*, thus establishing the default time zone for all commands that follow.

NOTE

Although **/etc/rc2** and **/etc/rc3** are ASCII text commands files, they are not meant to be "configurable". System services are started via procedures in **/etc/rc[23].d/S*** files. These files check for the presence of files in the **/etc/rcopts** directory. Thus, configurability is provided at the level of an "rcopt": that is, the system administrator creates an rcopt file with a specific name [see the README file in that directory], and pre-defined, non-configurable scripts perform all the necessary start and stop procedures. Customizations may also be added by creating files in **/etc/init.d** and making links to **/etc/rc[23].d** files.

FILES

/etc/init.d/* procedures to be started and stopped when there is a change in run-level; linked to files in **/etc/rc[23].d**; whether the file is executed at a particular run-level is determined by whether there is an S (start) file or K (stop) file in the **/etc/rc[23].d** directory that corresponds to the run-level

/etc/rc[23].d/S*
: files executed in ASCII order when **/etc/rc[23]** is called; each script starts a particular system service or performs some other startup function

/etc/rcopts/* their presence (and in some cases their contents) checked by some of the **/etc/rc[23].d/*** scripts to determine what system service (such as the spooler) must be started or stopped or other function performed.

Here are some hints about files in **/etc/rc[23].d**:

The order in which files are executed is important. Since they are executed in ASCII sort-sequence order, using the first character of the file name as a sequence indicator will help keep the proper order. Thus, files starting with the following characters would be:

[0-9]. very early
[A-Z]. early
[a-n]. later
[o-z]. last

Files in **/etc/rc[23].d** that begin with a dot (.) will not be executed. This feature can be used to hide files that are not to be executed for the time being without removing them.

Files in **/etc/rc2.d** must begin with an **S** or a **K** followed by a number and the rest of the file name. Upon entering run level 2, files beginning with **S** are executed with the **start** option; files beginning with **K**, are executed with the **stop** option. (The corresponding is true for run level 3 and files in **/etc/rc3.d**). Files beginning with other characters are ignored.

SEE ALSO

shutdown(1M).
S/Series CTIX Administrator's Guide.

NAME

rcmd - remote shell command execution

SYNOPSIS

/usr/local/bin/rcmd node [**-l** user] [**-n**] [command]

/usr/hosts/node [**-l** user] [**-n**] [command]

DESCRIPTION

The *rcmd* command sends *command* to *node* for execution. It passes the resulting remote command its own standard input and outputs the remote command's standard output and standard error. *Command* can consist of more than one parameter. The second, simplified form of the command is equivalent to the first, but is only available if the system administrator previously ran *mkhosts*(1M). Interrupt, quit, and terminate signals received by *rcmd* are also received by the remote command; *rcmd* normally terminates at the same time as the remote command.

If *command* is omitted, *rcmd* simply runs *rlogin*(1).

By default, the command belongs to the user on the remote node with the same name as the user who ran *rcmd*. This means that the resulting processes belong to the remote user and begin with the remote user's home directory as their working directory. Options permit you to specify another user on *node* as the owner. In any case, the remote system must have declared the local user equivalent to the remote user: an entry in **/etc/hosts.equiv** or in a **.rhosts** file in the current directory (normally the home directory) of the target user will demonstrate equivalence. [See *rcmd*(3).]

Options to *rcmd* follow:

-l *user* The command is to belong to *user* on *node*.

-n Prevent the remote command from blocking on input by making its standard input be **/dev/null** instead of *rcmd's* standard input.

If **-n** is not specified, *rcmd* reads the local standard input regardless of whether the remote machine reads standard input.

EXAMPLES

The following command runs **who** on a node called "central," putting the output in a file on the local machine:

```
rcmd central who > /tmp/c.who
```

The next example puts the same output on the remote machine:

```
rcmd central who \> /tmp/c.who
```

FILES

$HOME/.rhosts	(on the target machine)
/etc/hosts.equiv	(on the target machine)

SEE ALSO

rlogin(1), rshd(1M), rhosts(4).

REQUIREMENTS

rshd(1M) must be running on the target machine.

NOTE

In some installations, this command is called **rsh**, so as to be like other versions of the software.

WARNINGS

As the above examples illustrate, metacharacters to be interpreted by the remote shell must be hidden from the local shell. Thus

```
rcmd central cd /etc ; cat passwd
```

clearly doesn't do what was intended because the semicolon is interpreted by the local shell, not the remote shell, and the remote shell never even sees the **cat** command. Either of the following commands properly escapes the semicolon:

```
rcmd central cd /etc \; cat passwd
rcmd central 'cd /etc ; cat passwd'
```

NAME

rcp - remote file copy

SYNOPSIS

/usr/local/bin/rcp [**-r**] [**-p**] file1 [file2 ...] target

DESCRIPTION

The *rcp* command copies files between two nodes; working like the *cp* command [see *cp*(1)], with some extensions.

File1 is copied to *target*. If *target* is a directory, one or more files are copied into that directory; the copies have the same names as the originals.

File and directory names follow a convention that is an extension of the normal CTIX convention. Names take one of three forms:

user **@** *host* **:** *path*
host **:** *path*
path

where

host is the name of the system that contains or will contain the file. If no host is specified (the simple *path* form of the name), the system on which the command is executed is assumed.

user is the name of a user on the specified system. If no user is specified (the *host*:*path* and *path* forms of the name), the user name on the remote system that is the same as the user who executed the *rcp* command is used.

Access to the file system is as if by the specified user who has just logged in. Created files belong to the specified user and the specified user's group (taken from the password file). File and directory modifications can only occur if the specified user has permission to do them. If *path* does not begin with a slash (/), it is assumed to be relative to the specified user's home directory.

To use a user name on a remote system, the remote system must have declared it "equivalent" to your user name; see *rhosts*(4).

path is a conventional CTIX/UNIX path name. It can include file name generation sequences (*, ?, [...]); it may be necessary to quote the sequences to prevent their expansion on the local system.

An exclamation point (!) is allowed in place of the colon.

The **-r** (recursive) option copies directory hierarchies. If a file specified for copying is a directory and **-r** is specified, the entire hierarchy under it is copied. When **-r** is specified, *target* must be a directory.

When **-r** is not specified, copying directories is an error.

By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the *umask*(2) on the destination host is used. The **-p** option causes *rcp* to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the *umask*.

Note that a third system (not the source or target system of the copy) can execute *rcp*.

EXAMPLES

The following examples are executed on system alpha, by user fred. Alpha is networked to beta and gamma.

The first example copies **list** from fred's home directory on alpha to fred's home directory on beta:

```
rcp list beta:list
```

The next example copies a directory hierarchy. The original is rooted at **src** in fred's home directory on beta; the copy is to be rooted in **src** in the working directory:

```
rcp -r beta:src .
```

Finally, fred copies a file named **junk** from diane's home directory on beta to **/usr/tmp** on gamma; the copy on gamma is to belong to karl. Both diane and karl must have previously declared fred on alpha equivalent to their own user names [see *rhosts*(4)].

```
rcp diane\@beta:junk karl@gamma:/usr/tmp
```

Note that **junk** is not placed in karl's home directory because the *path* part of the name begins with a slash.

FILES

/etc/hosts.equiv
$HOME/.rhosts

SEE ALSO

hostname(1M), uname(1).

REQUIREMENTS

Both nodes involved in the copy must be running the *rshd*(1M) server.

DIAGNOSTICS

Most diagnostics are self-explanatory. ''Permission denied'' means either that the remote user does not have permission to do what you want or that the remote user is not equivalent to you.

WARNINGS

If a remote shell invoked by *rcp* has output on startup, *rcp* gets confused; *sh*(1) never has this problem, because it is not called as a login shell.

The **-r** option doesn't work correctly if the copy is purely local. Use *cpio*(1) instead.

NAME

reboot - reboot the system

SYNOPSIS

/etc/reboot [**-r** | **-h**]

DESCRIPTION

The *reboot* command issues a *uadmin*(2) call to unmount the **root** file system. When the **-r** option is used (this is the default), the system waits for the disks to become quiescent before rebooting. When the **-h** option is used, the system waits for the disks to become quiescent, and then halts in preparation for power off.

Note that only the super-user can execute *reboot*.

SEE ALSO

shutdown(1M).

NAME

regcmp - regular expression compile

SYNOPSIS

regcmp [-] files

DESCRIPTION

The *regcmp* command performs a function similar to *regcmp*(3X) and, in most cases, precludes the need for calling *regcmp*(3X) from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file*.**i**. If the - option is used, the output will be placed in *file*.**c**. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *File*.**i** files may thus be *included* in C programs, or *file*.**c** files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex*(*abc*,*line*) will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

EXAMPLES

```
name    "([A-Za-z][A-Za-z0-9_]*)$0"
telno   "\{0,1}([2-9][01][1-9])$0\){0,1} *"
        "([2-9][0-9]{2})$1[ -]{0,1}"
        "([0-9]{4})$2"
```

In the C program that uses the *regcmp* output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

SEE ALSO

regcmp(3X).

NAME

renice - alter priority of running process by changing nice

SYNOPSIS

/usr/local/bin/renice pid [priority]

DESCRIPTION

renice can be used by the super-user to alter the priority of a running process. By default, the nice of the process is made 19, which means that it will run only when nothing else in the system wants to. This can be used to lower the priority of long running processes that are interfering with interactive work.

renice can be given a second argument to choose a nice other than the default. A negative nice value can be used to raise the priority of a process.

FILES

/unix
/dev/kmem

SEE ALSO

nice(1).

BUGS

If you make the nice very negative, then the process cannot be interrupted. To regain control you must put the nice back (for example, to 0).

NAME

rexecd - remote execution server

SYNOPSIS

/etc/rexecd

DESCRIPTION

rexecd is the server for the *rexec*(3X) routine. The server provides remote execution facilities with authentication based on user names and encrypted passwords.

rexecd listens for service requests at the port indicated in the "exec" service specification; see *services*(4). When a service request is received the following protocol is initiated:

1) The server reads characters from the socket up to a null (0) byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is nonzero, it is interpreted as the port number of a secondary stream to be used for the *stderr*. A second connection is then created to the specified port on the client's machine.

3) A null-terminated user name of at most 16 characters is retrieved on the initial socket.

4) A null-terminated, encrypted, password of at most 16 characters is retrieved on the initial socket.

5) A null-terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

6) *rexecd* then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory and establishes the user and group protections of the user. If any of these steps fail, the connection is aborted with a diagnostic message returned.

7) A null byte is returned on the connection associated with the *stderr* and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rexecd*.

rexecd is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

SEE ALSO

inetd(1M), rexec(3X), inetd.conf(4), services(4).

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the *stderr*, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

"username too long"
The name is longer than 16 characters.

"password too long"
The password is longer than 16 characters.

"command too long"
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"Password incorrect."
The wrong password was supplied.

"No remote directory."
The *chdir* command to the home directory failed.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

BUGS

Indicating "Login incorrect" as opposed to "Password incorrect" is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rfadmin - Remote File Sharing administration

SYNOPSIS

rfadmin

rfadmin -[**ar**] *domain.nodename*

rfadmin -[**pq**]

rfadmin **-o** *option*

DESCRIPTION

The *rfadmin* command is primarily used to add and remove computers and their associated authentication information from a *domain*/**passwd** file on a Remote File Sharing primary domain name server. It is also used to transfer domain name server responsibilities from one machine to another. Used with no options, *rfadmin* returns the *domain.nodename* of the current domain name server for the local domain. Other options let you check if RFS is running and turn on the RFS loop back feature.

rfadmin can only be used to modify domain files on the primary domain name server (**-a** and **-r** options). If domain name server reponsibilities are temporarily passed to a secondary domain name server, that computer can use the **-p** option to pass domain name server responsibility back to the primary. *rfadmin* can be used on any computer with no options or with the **q** or **o** options. to print information about the current domain name server. The user must have **root** permissions to use the command.

-a *domain.nodename*

Used to add a computer to the member list of the domain that is served by this primary domain name server. The computer's name must be of the form *domain.nodename*. This command creates an entry for *nodename* in the *domain*/**passwd** file, which has the same format as /**etc/passwd**, and prompts for an initial authentication password. The password prompting process conforms with that of *passwd*(1).

-r *domain.nodename*

Used to remove a computer from its domain by removing it from the *domain*/**passwd** file.

-p Used to pass the domain name server responsibilities back to a primary or to a secondary name server.

-q Prints a message that will tell you whether or not RFS is running.

-o *option* Lets you set RFS system options, by replacing *option* with one of the following:

loopback
Enables loopback facility for your computer. When this is set, you can mount a resource that is advertised from your own computer. This is used for testing applications in RFS when only one computer is available. Loopback is disabled by default.

noloopback
Disables the loopback facility for your computer. This is the default.

ERRORS

When used with the **-a** option, if *domain.nodename* is not unique in the domain, an error message will be sent to standard error.

When used with the **-r** option, if (1) *domain.nodename* does not exist in the domain, (2) *domain.nodename* is defined as a domain name server, or (3) there are resources advertised by *domain.nodename*, an error message will be sent to standard error.

When used with the **-p** option to change the domain name server, if there are no backup name servers defined for *domain*, a warning message will be sent to standard error.

FILES

/usr/nserve/auth.info/*domain*/passwd

(For each *domain*, this file: is created on the primary, should be copied to all secondaries, and should be copied to all computers that want to do password verification of computers in the *domain*.)

SEE ALSO

passwd(1), rfstart(1M), rfstop(1M), umount(1M).
S/Series CTIX Administrator's Guide.

NAME

rfpasswd - change Remote File Sharing host password

SYNOPSIS

rfpasswd

DESCRIPTION

rfpasswd updates the Remote File Sharing authentication password for a host; processing of the new password follows the same criteria as *passwd*(1). The updated password is registered at the domain name server (**/usr/nserve/auth.info/***domain***/passwd**) and replaces the password stored at the local host (**/usr/nserve/loc.passwd** file).

This command is restricted to the super-user.

NOTE: If you change your host password, make sure that hosts that validate your password are notified of this change. To receive the new password, hosts must obtain a copy of the *domain*/**passwd** file from the domain's primary name server. If this is not done, attempts to mount remote resources may fail!

ERRORS

If (1) the old password entered from this command does not match the existing password for this machine, (2) the two new passwords entered from this command do not match, (3) the new password does not satisfy the security criteria in *passwd*(1), (4) the domain name server does not know about this machine, or (5) the command is not run with super-user privileges, an error message will be sent to standard error. Also, Remote File Sharing must be running on your host and your domain's primary name server. A new password cannot be logged if a secondary is acting as the domain name server.

FILES

/usr/nserve/auth.info/*domain*/passwd
/usr/nserve/loc.passwd

SEE ALSO

passwd(1), rfstart(1M), rfadmin(1M).

NAME

rfstart - start Remote File Sharing

SYNOPSIS

rfstart [**-v**] [**-p** *primary_addr*]

DESCRIPTION

rfstart starts Remote File Sharing and defines an authentication level for incoming requests. [This command can be used only after the domain name server is set up and your computer's domain name and network specification has been defined using *dname*(1M).]

-v Specifies that verification of all clients is required in response to initial incoming mount requests; any host not in the file **/usr/nserve/auth.info/***domain*/passwd for the **domain** they belong to, will not be allowed to mount resources from your host. If **-v** is not specified, hosts named in *domain*/passwd will be verified, other hosts will be allowed to connect without verification.

-p *primary_addr*
Indicates the primary domain name server for your domain. *primary_addr* must be the network address of the primary name server for your domain. If the **-p** option is not specified, the address of the domain name server is taken from the **rfmaster** file. [See *rfmaster*(1M) for a description of the valid address syntax.]

If the host password has not been set, *rfstart* will prompt for a password; the password prompting process must match the password entered for your machine at the primary domain name server [see *rfadmin*(1M)]. If you remove the **loc.passwd** file or change domains, you will also have to reenter the password.

Also, when *rfstart* is run on a domain name server, entries in the *rfmaster*(4) file are syntactically validated.

This command is restricted to the super-user.

ERRORS

If syntax errors are found in validating the *rfmaster*(4) file, a warning describing each error will be sent to standard error.

If (1) the shared resource environment is already running, (2) there is no communications network, (3) the domain name server cannot be found, (4) the domain name server does not recognize the machine, or (5) the command is run without super-user privileges, an error message will be sent to standard error.

Remote file sharing will not start if the host password in **/usr/nserve/loc.passwd** is corrupted. If you suspect this has happened, remove the file and run *rfstart* again to reenter your password.

NOTE: *rfstart* will NOT fail if your host password does not match the password on the domain name server. You will simply receive a warning message. However, if you try to mount a resource from the primary or any other host that validates your password, the mount will fail if your password does not match the one that host has listed for your machine.

FILES

/usr/nserve/rfmaster
/usr/nserve/loc.passwd

SEE ALSO

adv(1M), dname(1M), mount(1M), rfadmin(1M), rfstop(1M), unadv(1M), rfmaster(4).
S/Series CTIX Administrator's Guide.

NAME

rfstop - stop the Remote File Sharing environment

SYNOPSIS

rfstop

DESCRIPTION

rfstop disconnects a host from the Remote File Sharing environment until another *rfstart*(1M) is executed.

When executed on the domain name server, the domain name server responsibility is moved to a secondary name server as designated in the *rfmaster*(4) file. If there is no designated secondary name server **rfstop** will issue a warning message, Remote File Sharing will be stopped, and name service will no longer be available to the domain.

This command is restricted to the super-user.

ERRORS

If (1) there are resources currently advertised by this host, (2) resources from this machine are still remotely mounted by other hosts, (3) there are still remotely mounted resources in the local file system tree, (4) *rfstart*(1M) had not previously been executed, or (5) the command is not run with super-user privileges, an error message will be sent to standard error and Remote File Sharing will not be stopped.

SEE ALSO

adv(1M), mount(1M), rfadmin(1M), rfstart(1M), unadv(1M), rfmaster(4).
S/Series CTIX Administrator's Guide.

NAME

rfuadmin - Remote File Sharing notification shell script

SYNOPSIS

rfuadmin message remote_resource [seconds]

DESCRIPTION

The *rfuadmin* administrative shell script responds to unexpected Remote File Sharing events, such as broken network connections and forced unmounts, picked up by the *rfudaemon* process. This command is not intended to be run directly from the shell.

The response to messages received by *rfudaemon* can be tailored to suit the particular system by editing the *rfuadmin* script. The following paragraphs describe the arguments passed to *rfuadmin* and the responses.

disconnect *remote_resource*

A link to a remote resource has been cut. *rfudaemon* executes *rfuadmin*, passing it the message **disconnect** and the name of the disconnected resource. *rfuadmin* sends this message to all terminals using *wall*(1):

Remote_resource has been disconnected from the system.

Then it executes *fuser*(1M) to kill all processes using the resource, unmounts the resource [*umount*(1M)] to clean up the kernel, and starts *rmount* to try to remount the resource.

fumount *remote_resource*

A remote server machine has forced an unmount of a resource a local machine has mounted. The processing is similar to processing for a disconnect.

fuwarn *remote_resource seconds*

This message notifies *rfuadmin* that a resource is about to be unmounted. *rfudaemon* sends this script the *fuwarn* message, the resource name, and the number of seconds in which the forced unmount will occur. *rfuadmin* sends this message to all terminals:

Remote_resource is being removed from the system in # seconds.

SEE ALSO

fumount(1M), rmount(1M), rfudaemon(1M), rfstart(1M), wall(1).

NAME

rfudaemon - Remote File Sharing daemon process

SYNOPSIS

rfudaemon

DESCRIPTION

The *rfudaemon* command is started automatically by *rfstart*(1M) and runs as a daemon process as long as Remote File Sharing is active. Its function is to listen for unexpected events, such as broken network connections and forced unmounts, and execute appropriate administrative procedures.

When such an event occurs, *rfudaemon* executes the administrative shell script *rfuadmin*, with arguments that identify the event. This command is not intended to be run from the shell. Here are the events:

DISCONNECT

A link to a remote resource has been cut. *rfudaemon* executes *rfuadmin*, with two arguments: **disconnect** and the name of the disconnected resource.

FUMOUNT

A remote server machine has forced an unmount of a resource a local machine has mounted. *rfudaemon* executes *rfuadmin*, with two arguments: *fumount* and the name of the disconnected resource.

GETUMSG

A remote user-level program has sent a message to the local *rfudaemon*. Currently the only message sent is *fuwarn*, which notifies *rfuadmin* that a resource is about to be unmounted. It sends *rfuadmin* the *fuwarn*, the resource name, and the number of seconds in which the forced unmount will occur.

LASTUMSG

The local machine wants to stop the *rfudaemon* [*rfstop*(1M)]. This causes *rfudaemon* to exit.

SEE ALSO

rfstart(1M), rfuadmin(1M).

NAME

riopcfg - configure system for Remote I/O Processor

SYNOPSIS

/etc/riop/riopcfg [options] [**-r** release] [riopnumber ...]

DESCRIPTION

The *riopcfg* command is used to configure and give status information about Remote I/O Processors (RIOPs). An RIOP number is a decimal number in the range 0 to 31 which is used to order all RIOPs in a system. This number is placed in the second field of the **/etc/riop/rtab** file when an RIOP is put in service [see *rtab*(4)]. If one or more RIOP numbers are passed as arguments, the options are processed for those RIOP numbers only. If no RIOP number is given, the options are processed for all RIOP numbers listed in *rtab*. If no RIOP number is given and *rtab* does not exist, *riopcfg* assumes that an initial install is being done, prompts for the number of RIOPs to be supported, and creates a template *rtab*, which it then uses as a guide to process the specified options.

The following options are allowed:

-d Create the 16 tty devices associated with this RIOP. If the directory **/dev/rtty** exists, start naming for RIOP number 0 from **/dev/rtty/r000;** otherwise, start from **/dev/tty400.** The program complains if any of the device nodes already exist, but continues making the rest of the nodes.

-i Append entries to **/etc/inittab** for the 16 tty devices associated with this RIOP. Write entries with *getty*s set for 9600 BAUD and turned off. If any of the entries for an RIOP already exist, make no entries for that RIOP.

-s Print out the status of the RIOP. Output is in tabular form giving RIOP number, state, unique ID, line, drop, and number of ports.

-m Print out the range of tty devices supported by this RIOP.

-b Cause the RIOP to reboot. An I/O control call to the driver causes a reboot message to be sent to the RIOP.

-x Turn on external echo and output processing for this RIOP. All ports are set for external processing. This takes effect for each port during its next closed to open transition.

-n Turn off external echo and output processing for this RIOP. All ports are set for no external processing. This takes effect for each port during its next closed to open transition.

-r The string following this option is taken as the release level to be used in filling in the third field of **/etc/riop/rtab** when a template file is needed during installation.

DIAGNOSTICS

The program prints out informational messages describing the results of its processing.

FILES

/etc/riop/rtab

SEE ALSO

extproc(1M), rtab(4).

NAME

riopqry - query Remote I/O Processor for online data

SYNOPSIS

/etc/riop/riopqry [**-p**] address length [riopnumber ...]

DESCRIPTION

The *riopqry* command queries for and displays *length* memory data bytes beginning at *address* from online Remote I/O Processors (RIOPs). An RIOP number is a decimal number in the range 0 to 31 which is used to order all RIOPs in a system. This number is placed in the second field of the **/etc/riop/rtab** file when an RIOP is put in service [see *rtab*(4)]. If one or more RIOP numbers are passed as arguments, each RIOP is queried in turn. If no RIOP number is given then each RIOP number listed in *rtab* is specified.

The **-p** option causes the retrieved data to be printed as an ASCII hexadecimal dump. Without this option the binary data is sent to the standard output. When multiple RIOPs are specified, an RIOP identification line is printed to standard error before each query action.

FILES

/etc/riop/rtab

SEE ALSO

rtab(4)

NAME

rlogin - remote login

SYNOPSIS

/usr/local/bin/rlogin host [**-ec**] [**-l** name]
/usr/hosts/host [**-ec**] [**-l** name]

DESCRIPTION

The *rlogin* command connects you to a login shell executing on *host*. The second simplified form of the command is equivalent to the first, but is available only if *mkhosts*(1M) was previously run by the system administrator. By default *rlogin* uses the same user name on the remote host that the user is using on the local host. The remote login program does not require a password if the remote host has declared the two users equivalent [see *rhosts*(4)].

The *rlogin* command attempts to configure the remote "terminal" in a convenient way. The TERM environment variable on the remote shell is automatically set to match its value on the local shell which ran *rlogin*. Echoing takes place at the remote host. Flow control on XON/XOFF and flushing of input and output on interrupts are handled properly.

Close the connection by hanging up on *rlogin*, by logging out of the remote host, or by typing "~." (tilde-period) at the beginning of a line. The hangup and the tilde-period command both cause a hangup on the remote "terminal." To send an input line beginning with tilde to the remote host, begin the line with two tildes.

The *rlogin* command understands the following options:

-e*c* Use the character *c* instead of tilde as the escape character. There must not be a space between **e** and *c* on the command line. A **c.** (c-period) at the beginning of an input line closes the connection, and *cc* at the beginning of an input line sends a single *c*.

-l *user* Login as *user* on the remote system. *User's* password is not required provided that the local user name is on *user's* list of "equivalent" user names. See *rhosts*(4).

SEE ALSO

rcmd(1), rlogind(1M), rhosts(4).

NAME

rlogind - remote login server

SYNOPSIS

/etc/rlogind

DESCRIPTION

The *rlogind* network server supports remote logins by programs such as *rlogin*(1). It is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

The *rlogind* server enforces an authentication procedure based on equivalence of user names [see *rhosts*(4)]. This procedure assumes all hosts on the network are equally secure.

SEE ALSO

inetd(1M), rlogin(1), inetd.conf(4), rhosts(4), services(4).

NAME

rm, rmdir - remove files or directories

SYNOPSIS

rm [**-f**] [**-i**] file ...

rm -r [**-f**] [**-i**] dirname ... [file ...]

rmdir [**-p**] [**-s**] dirname ...

DESCRIPTION

The *rm* command removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with **y** (for yes), the file is deleted, otherwise the file remains.

Note that if the standard input is not a terminal, the command will operate as if the **-f** option is in effect.

rmdir removes the named directories, which must be empty.

Three options apply to *rm*:

-f This option causes the removal of all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory was attempted, this option cannot suppress an error message.

-r This option causes the recursive removal of any directories and subdirectories in the argument list. The directory will be emptied of files and removed. Note that the user is normally prompted for removal of any write-protected files which the directory contains. The write-protected files are removed without prompting, however, if the **-f** option is used, or if the standard input is not a terminal and the **-i** option is not used.

If the removal of a non-empty, write-protected directory was attempted, the command will always fail (even if the **-f** option is used), resulting in an error message.

-i With this option, confirmation of removal of any write-protected file occurs interactively. It overrides the **-f** option and remains in effect even if the standard input is not a terminal.

Two options apply to *rmdir*:

-p This option allows users to remove the directory *dirname* and its parent directories which become empty. A message is printed on standard output as to whether the whole path is removed or part of the path remains for some reason.

-s This option is used to suppress the message printed on standard output when **-p** is in effect.

DIAGNOSTICS

All messages are generally self-explanatory.
To avoid the consequences of inadvertently doing something like the following, the files . and .. cannot be removed.

```
rm -r .*
```

Both *rm* and *rmdir* return exit codes of 0 if all the specified directories are removed successfully. Otherwise, they return a non-zero exit code.

SEE ALSO

unlink(2), rmdir(2).

NAME

rmdel - remove a delta from an SCCS file

SYNOPSIS

rmdel -r*SID* files

DESCRIPTION

rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the *SID* specified must *not* be that of a version being edited for the purpose of making a delta (that is, if a *p-file* [see *get*(1)] exists for the named SCCS file, the *SID* specified must *not* appear in any entry of the *p-file*).

The **-r** option is used for specifying the *SID* (**SCCS IDentification**) level of the delta to be removed.

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

Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES

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

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

rmntstat - display mounted resource information

SYNOPSIS

rmntstat [**-h**] [*resource*]

DESCRIPTION

When used with no options, *rmntstat* displays a list of all local Remote File Sharing resources that are remotely mounted, the local path name, and the corresponding clients. *rmntstat* returns the remote mount data regardless of whether a resource is currently advertised; this ensures that resources that have been unadvertised but are still remotely mounted are included in the report. When a *resource* is specified, *rmntstat* displays the remote mount information only for that resource. The **-h** option causes header information to be omitted from the display.

EXIT STATUS

If no local resources are remotely mounted, *rmntstat* will return a successful exit status.

ERRORS

If *resource* (1) does not physically reside on the local machine or (2) is an invalid resource name, an error message will be sent to standard error.

SEE ALSO

mount(1M), fumount(1M), unadv(1M).

NAME

rmnttry - attempt to mount remote resources

SYNOPSIS

/usr/nserve/rmnttry [resource ...]

DESCRIPTION

The *rmnttry* command sequences through the pending mount requests stored in **/usr/nserve/rmnttab**, trying to mount each resource. If a mount succeeds, the resource entry is removed from the **/usr/nserve/rmnttab** file.

If specific (one or more) resource names are supplied, mounts are attempted only for those resources, rather than for all pending mounts. Mounts are not attempted for resources not present in the **/usr/nserve/rmnttab** file [see *rmount*(1M)]. If a mount invoked from *rmnttry* takes over three minutes to complete, *rmnttry* aborts the mount and issues a warning message.

The *rmnttry* command is typically invoked from a *cron* entry in **/u/spool/cron/crontabs/root** to attempt mounting queued resources at periodic intervals. The default strategy is to attempt mounts at 15-minute intervals, as shown in the following *cron* entry:

```
10,24,40,55 * * * * /usr/nserve/rmnttry > /dev/null
```

FILES

/usr/nserve/rmnttab pending mount requests

SEE ALSO

cron(1M), mount(1M), rmount(1M), rumount(1M), mnttab(4).

DIAGNOSTICS

The *rmnttry* command returns the following codes:

0 Success.

1 One or more mounts failed.

2 Incorrect usage.

NAME

rmount - queue remote resource mounts

SYNOPSIS

/etc/rmount [**-d**[**r**] *resource directory*]

DESCRIPTION

The *rmount* command queues a remote resource for mounting. The command enters the resource request into **/usr/nserve/rmnttab**, which is formatted identically to **/etc/mnttab**. The *rmnttry*(1M) command is used to poll entries in the **rmnttab** file.

When used without arguments, **rmount** prints a list of resources with pending mounts, along with their destined directories, modes, and dates of request. The resources are listed chronologically, the oldest resource request first.

The following options are available:

-d Indicates that the *resource* is a remote resource to be mounted on *directory*.

-r Indicates that the *resource* is to be mounted read-only. If the *resource* is write-protected, this flag must be used.

FILES

/usr/nserve/rmnttab pending mount requests

SEE ALSO

mount(1M), rmnttry(1M), rumount(1M), rmountall(1M), mnttab(4).

DIAGNOSTICS

An exit code of 0 is returned upon successful completion of *rmount*; otherwise, a non-zero value is returned.

NAME

rmountall, rumountall - mount, unmount Remote File Sharing (RFS) resources

SYNOPSIS

/etc/rmountall [-] " file-system-table " [...]

/etc/rumountall [**-k**]

DESCRIPTION

The *rmountall* Remote File Sharing (RFS) command is used to mount remote resources according to a *file-system-table.* (Note that **/etc/fstab** is the recommended *file-system-table.*) The special file name dash (-) reads from the standard input. The *rmountall* command also invokes the **/usr/nserve/rmnttry**(1M) command, which attempts to mount queued resources.

The *rumountall* command causes all mounted remote resources to be unmounted and deletes all resources that were queued from **rmount**(1M). The **-k** option sends a SIGKILL signal, through *fuser*(1M), to processes that have files open.

These commands can be executed only by the super-user.

The file-system-table format is as follows:

column 1	block special file name of file system
column 2	mount-point directory
column 3	**-r** if to be mounted read-only; **-d** if remote resource
column 4	file system type (not used with Remote File Sharing)
column 5+	ignored

White-space separates columns. Lines beginning with a pound sign (#) are comments. Empty lines are ignored.

SEE ALSO

fuser(1M), mount(1M), rfstart(1M), rmnttry(1M), rmount(1M), signal(2).

DIAGNOSTICS

No messages are printed if the remote resources are mounted successfully.

Error and warning messages come from *mount*(1M).

NAME

route - manually manipulate the routing tables

SYNOPSIS

/etc/route [**-f**] [command destination gateway [metric]]

DESCRIPTION

route is a program used to manually manipulate the network routing tables. It is normally not needed, since the routing daemon, *routed* manages the system routing table and therefore handles this function.

route accepts two commands: *add*, to add a route; and *delete*, to delete a route.

All commands have the following syntax:

/etc/**route** *command destination gateway* [*metric*]

where *destination* is a host or network for which the route is "to", *gateway* is the gateway to which packets should be addressed, and *metric* is an optional count indicating the number of hops to the *destination*. If no metric is specified, *route* assumes a value of 0. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. If the *destination* has a "local address part" of INADDR_ANY, the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. *NOTE*: If the route is to a destination connected via a gateway, *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first in the host name database; see *hosts*(4). If this lookup fails, the name is then looked for in the network name database; see *networks*(4).

route uses a raw socket and the SIOCADDRT and SIOCDELRT *ioctl*'s to do its work. As such, only the super-user may modify the routing tables.

If the **-f** option is specified, *route* will "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

DIAGNOSTICS

add host*: gateway* host *flags* hex-flag

The specified route is being added to the tables. The values printed are from the routing table entry supplied in the *ioctl* call.

delete host*: gateway* host *flags* hex-flags

As above, but when deleting an entry.

host host *done*

When the **-f** flag is specified, each routing table entry deleted is indicated with a message of this form.

A delete operation was attempted for an entry which
wasn't present in the tables.

An add operation was attempted, but the system was
low on resources and was unable to allocate memory to create the new entry.

SEE ALSO

intro(4), adman(1), routed(1M), hosts(4), networks(4).

NAME

routed - network routing daemon

SYNOPSIS

/etc/routed [**-d**] [**-g**] [**-s**] [**-t**] [logfile]

/etc/routed [**-d**] [**-g**] [**-q**] [**-t**] [logfile]

DESCRIPTION

The *routed* daemon manages the Internet routing tables by using a variant of the Xerox NS Routing Information Protocol. It is invoked at boot time and should be started if the file **/etc/rcopts/KINET** is present.

In normal operation *routed* listens on the *udp*(7) socket for the *route* service [see *services*(4)] for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When *routed* is started, it uses the SIOCGIFCONF *ioctl* to find those directly connected interfaces configured into the system and marked "up" (the software loopback interface is ignored). If multiple interfaces are present, it is assumed that the host will forward packets between networks. The *routed* daemon then transmits a *request* packet on each interface (by using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, *routed* formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a "hop count" metric (a count of 16, or greater, is considered "infinite"). The metric associated with each route returned provides a metric *relative to the sender*.

Response packets received by *routed* are used to update the routing tables if one of the following conditions is satisfied:

(1) No routing table entry exists for the destination network or host, and the metric indicates the destination is "reachable" (for example, the hop count is not infinite).

(2) The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.

(3) The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.

(4) The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, *routed* records the change in its internal tables and updates the kernel routing table. The change is reflected in the next *response* packet sent.

In addition to processing incoming packets, *routed* also periodically checks the routing table entries. If an entry has not been updated for three minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to ensure that the invalidation is propagated throughout the local internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly-connected hosts and networks. The response is sent to the broadcast address on networks capable of that function, to the destination address on point-to-point links, and to the router's own address on other networks. The normal routing tables are bypassed when sending gratuitous responses. The reception of responses on each network is used to determine that the network and interface are functioning correctly. If no response is received on an interface, another route may be chosen to route around the interface, or the route may be dropped if no alternative is available.

The *routed* daemon supports several options:

-d Enable additional debugging information to be logged, such as bad packets received.

-g Used on internetwork routers to offer a route to the "default" destination. This is typically used on a gateway to the Internet, or on a gateway that uses another routing protocol whose routes are not reported to other local routers.

-s Forces *routed* to supply routing information whether it is acting as an internetwork router or not. This is the default if multiple network interfaces are present, or if a point-to-point link is in use.

-q The opposite of the **-s** option.

-t All packets sent or received are printed on the standard output. In addition, *routed* does not divorce itself from the controlling terminal so that interrupts from the keyboard kill the process.

Any other argument supplied is interpreted as the name of file in which *routed*'s actions should be logged. This log contains information about any

changes to the routing tables and, if not tracing all packets, a history of recent messages sent and received which are related to the changed route.

In addition to the *routed* facilities, routed also supports the notion of *distant* gateways. When *routed* is started up it reads the file *etc/gateways* to find gateways that may not be located using only information from the SIOGIFCONF *ioctl*.

FILES

/etc/gateways for distant gateways

SEE ALSO

"Internet Transport Protocols," XSIS 028112, Xerox System Integration Standard.
udp(7), gateways(4).

BUGS

The kernel's ICMP routing tables may not correspond to those of *routed* when redirects change or add routes.

NAME

rpcinfo - report RPC information

SYNOPSIS

rpcinfo -p [host]
rpcinfo -u host program-number [version-number]
rpcinfo -t host program-number [version-number]

DESCRIPTION

The *rpcinfo* command makes an RPC call to an RPC server and reports what it finds.

OPTIONS

-p Probe the portmapper on *host*, and print a list of all registered RPC programs. If *host* is not specified, it defaults to the hostname returned by *hostname*(1).

-u Make an RPC call to procedure 0 of *program-number* using UDP, and report whether a response was received.

-t Make an RPC call to procedure 0 of *program-number* using TCP, and report whether a response was received.

The *program-number* argument can be either a name or a number. If no version is given, it defaults to 1.

FILES

/etc/rpc names for RPC program numbers

SEE ALSO

rpc(4), portmap(1M).

NAME

rshd - remote shell server

SYNOPSIS

/etc/rshd

DESCRIPTION

rshd is the network server for programs such as *rcmd*(1) and *rcp*(1) which need to execute a noninteractive shell on remote machines. *rshd* is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

rshd enforces an authentication procedure based on equivalence of user names [see *rhosts*(4)]. This procedure assumes all nodes on the network are equally secure.

SEE ALSO

inetd(1M), rcmd(1), rcp(1), inetd.conf(4), rhosts(4).

NAME

rsterm - manually start and stop terminal input and output

SYNOPSIS

/usr/local/bin/rsterm number device

DESCRIPTION

The *rsterm* command manually exercises the start/stop features of the terminal driver. [For a discussion of start/stop features, see the STOP and START characters and IXON, IXANY, and IXOFF flags under *termio*(7).] The *rsterm* command requires two parameters:

number A number specifying the action:

- *0* Suspend output as if the terminal had sent a STOP character to the system.
- *1* Resume output as if the terminal had sent a START character to the system.
- *2* Block input by sending the terminal a STOP character as if the terminal had nearly filled the terminal's input queue.
- *3* Unblock input by sending the terminal a START character as if the system had nearly emptied the terminal's input queue.

device The special file for the terminal.

Normally, STOP is the ASCII XOFF character, Control-S, and START is the ASCII XON character, Control-Q.

Operation 2 (resume output) is the most often-used parameter. Use it when a terminal (a printer for example) has sent a STOP character and cannot be made to send a START character.

The *rsterm* command provides one way to clear up a terminal. Another way is to kill all processes associated with the terminal: this momentarily closes the special file, returning all terminal modes to their initial state; see *kill*(1).

Note that you must be the super-user to run *rsterm*.

FILES

/dev/tty??? - terminal devices

SEE ALSO

kill(1), termio(7).

NAME

rtpenable - real-time priorities enabled/disabled

SYNOPSIS

/etc/rtpenable -e | d

DESCRIPTION

rtpenable is used to enable or disable the additional range of eight negative nice values (values in the range **-21** to **-28** [see *nice*(2)]). The additional range of nice values is fixed and pre-emptive. This range will be automatically enabled when the system boots if the zero length file **/etc/rcopts/RTPENABLE** exists. As distributed, CTIX has the extra range of priorities disabled.

The following options are recognized by *rtpenable*:

-e enable the additional priority range.

-d disable the additional priority range.

FILES

/etc/rcopts/RTPENABLE

SEE ALSO

nice(2), syslocal(2).

WARNING

Since the additional range of negative nice values are pre-emptive, they should be used with utmost care.

NAME

rumount - cancel queued remote resource request

SYNOPSIS

/usr/nserve/rumount resource ...

DESCRIPTION

The *rumount* command cancels a request for one or more resources that are queued for mount. Entries for the resources are deleted from **/usr/nserve/rmnttab**.

FILES

/usr/nserve/rmnttab pending mount requests

SEE ALSO

mount(1M), rmnttry(1M), rmount(1M), rmountall(1M), mnttab(4).

DIAGNOSTICS

The following exit codes are returned by *rumount*:

0 Successful.

1 Resource request for dequeing is not in **/usr/nserve/rmnttab**.

2 Bad usage or error in reading/writing **/usr/nserve/rmnttab**.

NAME

runacct - run daily accounting

SYNOPSIS

/usr/lib/acct/runacct [mmdd [state]]

DESCRIPTION

The **runacct** command invokes the main daily accounting shell procedure. It is normally initiated via *cron*(1M). *runacct* processes connect, fee, disk, and process accounting files. It also prepares summary files for *prdaily* or billing purposes. Disk block counts are reported for 512-byte blocks.

runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into **active**. When an error is detected, a message is written to **/dev/console**, mail[(see *mail*(1)] is sent to **root** and **adm**, and *runacct* terminates. *runacct* uses a series of lock files to protect against re-invocation. The files **lock** and **lock1** are used to prevent simultaneous invocation, and **lastdate** is used to prevent more than one invocation per day.

runacct breaks its processing into separate, restartable *states* using **statefile** to remember the last *state* completed. It accomplishes this by writing the *state* name into **statefile**. *runacct* then looks in **statefile** to see what it has done and to determine what to process next. *States* are executed in the following order:

SETUP	Move active accounting files into working files.
WTMPFIX	Verify integrity of **wtmp** file, correcting date changes if necessary.
CONNECT1	Produce connect session records in **ctmp.h** format.
CONNECT2	Convert **ctmp.h** records into **tacct.h** format.
PROCESS	Convert process accounting records into **tacct.h** format.
MERGE	Merge the connect and process accounting records.
FEES	Convert output of *chargefee* into **tacct.h** format and merge with connect and process accounting records.
DISK	Merge disk accounting records with connect, process, and fee accounting records.
MERGETACCT	Merge the daily total accounting records in **daytacct** with the summary total accounting records in **/usr/adm/acct/sum/tacct**.

CMS Produce command summaries.

USEREXIT Any installation-dependent accounting programs can be included here.

CLEANUP Cleanup temporary files and exit.

To restart *runacct* after a failure, first check the **active** file for diagnostics, then fix up any corrupted data files such as **pacct** or **wtmp**. The **lock** files and **lastdate** file must be removed before *runacct* can be restarted. The argument *mmdd* is necessary if *runacct* is being restarted, and specifies the month and day for which *runacct* will rerun the accounting. Entry point for processing is based on the contents of **statefile**; to override this, include the desired *state* on the command line to designate where processing should begin.

EXAMPLES

To start *runacct*:
nohup runacct 2> /usr/adm/acct/nite/fd2log &

To restart *runacct*:
nohup runacct 0601 2>> /usr/adm/acct/nite/fd2log &

To restart *runacct* at a specific *state*:
nohup runacct 0601 MERGE 2>> /usr/adm/acct/nite/fd2log &

FILES

/etc/wtmp
/usr/adm/pacct*
/usr/src/cmd/acct/tacct.h
/usr/src/cmd/acct/ctmp.h
/usr/adm/acct/nite/active
/usr/adm/acct/nite/daytacct
/usr/adm/acct/nite/lock
/usr/adm/acct/nite/lock1
/usr/adm/acct/nite/lastdate
/usr/adm/acct/nite/statefile
/usr/adm/acct/nite/ptacct*.*mmdd*

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), mail(1), acct(2), acct(4), utmp(4).
S/Series CTIX Administrator's Guide.

BUGS

Normally it is not a good idea to restart *runacct* in the **SETUP** *state*. Run **SETUP** manually and restart via:

runacct *mmdd* **WTMPFIX**

If *runacct* failed in the **PROCESS** *state*, remove the last **ptacct** file because it will not be complete.

NAME

ruptime - display status of nodes on local network

SYNOPSIS

/usr/local/bin/ruptime [**-a**] [**-l**] [**-t**] [**-u**]

DESCRIPTION

The *ruptime* command displays status information for hosts on the local network. For each node, a line reports: the node name; whether the node is up [a node is considered "down" if its *rwhod*(1M) server has not broadcast in five minutes]; the time the node has been up in days, hours, and minutes; the number of logged-in users logged who have used their keyboards in the last hour; and the load (average number of jobs in the run queue) for the last one minute, five minutes, and 15 minutes.

When no options are specified, the status lines are sorted by node name.

Options to *ruptime* follow:

-a Count all logged-in users, including idle ones.

-l Sort status lines by load average.

-t Sort status lines by time node has been up.

-u Sort status lines by number of users.

REQUIRMENTS

Each node to be listed must be running the *rwhod*(1M) server, which broadcasts a status packet once a minute. The local node must also be running this server to maintain data files.

FILES

/usr/spool/rwho/whod.* data files

SEE ALSO

rwho(1), rwhod(1M).

NAME

rwho - who is logged in on local network

SYNOPSIS

/usr/local/bin/rwho [**-a**]

DESCRIPTION

The *rwho* command lists users logged in on machines on the local network. The format is similar to that of *who*(1). Without options, only users who have typed in the last hour are listed. For each user listed, *rwho* displays the user name; the host name; and the date and time the user logged in. If the user has not typed in the last minute, *rwho* also displays the user's idle time in hours and minutes.

Options to *rwho* follow:

-a List all users on active hosts (users idle for more than an hour are listed).

If information from a host is more than five minutes old, the host is assumed to be down and its users are not listed.

REQUIREMENTS

Each host to be listed must be running the *rwhod*(1M) server, which broadcasts a status packet once a minute. The local host must also be running this server to maintain the data files. Since broadcasts do no cross gateways, hosts on other networks are not listed.

FILES

/usr/spool/rwho/whod.* information about other hosts

SEE ALSO

ruptime(1), rwhod(1M).

NAME

rwhod - host status server

SYNOPSIS

/etc/rwhod

DESCRIPTION

rwhod collects and distributes information about hosts on the local network, including the local host. It is normally started via an entry in **/etc/rcopts/NETD**. It performs four chores once a minute:

- Gathers information about the local host.
- Broadcasts information about the local host for the benefit of *rwhod* servers running on other hosts.
- Collects information broadcast by *rwhod* servers on other hosts.
- Maintains network status files, using information gathered by this and the other *rwhod* servers.

The files maintained by *rwho* have names of the form **/usr/spool/rwho/whod.***name*, where *name* is the name of the host whose status is in the file. Each status file begins with the header of the following form:

```
struct whod {
        char  wd_vers;              /* version number */
        char  wd_type;              /* type number */
        char  wd_fill;              /* ignored */
        int   wd_sendtime;          /* time this packet sent */
        int   wd_recvtime;          /* time this packet received */
        char  wd_hostname[32];      /* name of originating hosts */
        int   wd_loadav[3];         /* load averages; see rwho(1) */
        int   wd_boottime;          /* boot time of originating host */
};
```

The host name of a system is printed by the *uname*(1) command. The remainder of the file consists of user records:

```
struct outmp {
        char  out_line[8];      /* terminal name */
        char  out_name[8];      /* user name */
        int   out_ltime;        /* login time */
        int   out_itime;        /* idle time */
};
```

rwho performs an *nlist*(3C) on /**unix** every 10 minutes in case that file is not the current system image.

rwho transmits and receives messages at the port indicated in the "rwho" service specification. See *services*(4).

FILES

/usr/spool/rwho/whod.*
/etc/rcopts/NETD

SEE ALSO

rwho(1), ruptime(1).

WARNINGS

Death of this server makes other hosts think that this host is down.

NAME

sact - print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the **-e** option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

Field 1	specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	specifies the SID for the new delta to be created.
Field 3	contains the logname of the user who will make the delta (that is, executed a *get* for editing).
Field 4	contains the date that **get -e** was executed.
Field 5	contains the time that **get -e** was executed.

SEE ALSO

delta(1), get(1), unget(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sadp - disk access profiler

SYNOPSIS

sadp [**-th**] [**-d** disk[-drive]] s [n]

DESCRIPTION

The *sadp* program reports disk-access location and seek distance, in tabular or histogram form. It samples disk activity once every second during an interval of *s* seconds, repeatedly if *n* is specified. Cylinder usage and disk distance are recorded in units of 8 cylinders.

The valid value of *disk* is **disk**. *Drive* specifies the disk drives as one of the following:

- A drive number in the range supported by *disk* (currently, **0** through **255**).
- Two drive numbers separated by a minus (indicating an inclusive range).
- A list of drive numbers separated by commas.

Up to eight disk drives can be reported. If only one *disk* is present. the **-d** option can be omitted.

The **-t** flag (default) causes the data to be reported in tabular form. The **-h** flag produces a histogram on the printer of the data.

EXAMPLES

The following command generates four tabular reports, each describing cylinder usage and seek distance of disk drive 0 during a 15-minute interval:

sadp -d disk-0 900 4

The following command generates a histogram for drive 0 (c0d0) and drive 2 during a 5-minute interval:

sadp -h -d disk-0,2 300

FILES

/dev/kmem

SEE ALSO

S/Series CTIX Administrator's Guide.

NOTE

If the time inverval is too small, *sadp* produces an invalid report.

NAME

sag - system activity graph

SYNOPSIS

sag [options]

DESCRIPTION

The *sag* command graphically displays the system activity data stored in a binary data file by a previous *sar*(1) run. Any of the *sar* data items can be plotted singly, or in combination: as cross plots, or versus time. Simple arithmetic combinations of data can be specified. The *sag* command invokes *sar* and finds the desired data by string-matching the data column header [see *sar (1)* to see what is available]. The following *options* are passed through to *sar*:

-s *time* Select data later than *time* in the form hh [:mm]. Default is 08:00.

-e *time* Select data up to *time*. Default is 18:00.

-i *sec* Select data at intervals as close as possible to *sec* seconds.

-f *file* Use *file* as the data source for *sar*. Default is the current daily data file **/usr/adm/sa/sa***dd*.

Other *options*:

-T *term* Produce output suitable for terminal *term*. See *tplot*(1G) for known terminals. Default for *term* is **$TERM**.

-x *spec* x axis specification with *spec* in the following form:

name [op name] ... [lo hi]

-y *spec* y axis specification with *spec* in the same form as above.

Name is either a string that matches a column header in the *sar* report, with an optional device name in square brackets (for example, **r+w/s[dsk-1]**), or an integer value. *Op* is + - * or / surrounded by blanks. Up to five names can be specified. Parentheses are not recognized. Contrary to custom, + and - have precedence over * and /. Evaluation is left to right. Thus A / A + B * 100 is evaluated (A/(A+B))*100, and A + B / C + D is (A+B)/(C+D). *Lo* and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

A single *spec* is permitted for the x axis. If unspecified, *time* is used. Up to 5 *specs* separated by semicolons (;) can be given for **-y**. Enclose the **-x** and **-y** arguments in double quotation marks (" ") if blanks or \carriage return are included. The **-y** default is shown below:

-y "%usr 0 100; %usr + %sys 0 100; %usr +%sys + %wio 0 100"

EXAMPLES

The following command displays today's CPU utilization:

sag

The following command sequence reports on activity of all disk drives over a period of 15 minutes:

```
TS=date +%H:%M
sar -o tempfile 60 15
TE=date +%H:%M
sag -f tempfile -s $TS -e $TE -y "r+w/s[dsk]"
```

FILES

/usr/adm/sa/sa*dd* daily data file for day *dd*.

SEE ALSO

sar(1), tplot(1G).
S/Series CTIX Administrator's Guide.

NAME

sar - system activity reporter

SYNOPSIS

sar [**-ubdycwaqvmprDSAC**] [**-o** file] t [n]

sar [**-ubdycwaqvmprDSAC**] [**-s** time] [**-e** time] [**-i** sec] [**-f** file]

DESCRIPTION

In the first instance, *sar* samples cumulative activity counters in the operating system at *n* intervals of *t* seconds, where *t* should be 5 or greater. If the **-o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, **sar** extracts data from a previously recorded *file*, either the one specified by the **-f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd*. The starting and ending times of the report can be bounded by use of the **-s** and **-e** *time* arguments of the form *hh*[:*mm*[:*ss*]]. The **-i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by the following options:

-u Report CPU utilization (the default):

%usr, %sys, %wio, %idle
: Portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle. When used with **-D**, **%sys** is split into percentage of time servicing requests from remote machines (**%sys remote**) and all other system time (**%sys local**).

-b Report buffer activity:

bread/s, bwrit/s
: Transfers per second of data between system buffers and disk or other block devices.

lread/s, lwrit/s
: Accesses of system buffers.

%rcache, %wcache
: Cache-hit ratios: that is, (1-bread/lread) as a percentage.

pread/s, pwrit/s
: Transfers through raw (physical) device mechanism. When used with **-D**, buffer caching is reported for locally-mounted remote resources.

-d Report activity for each block device: for example, disk or tape drive. When data is displayed, the device specification *dsk-* is generally used to represent a disk drive. The following activity data is reported:

%busy, avque Portion of time device was busy servicing a transfer request, average number of requests outstanding during that time.

r+w/s, blks/s Number of data transfers from or to device, number of bytes transferred in 512-byte units.

avwait, avserv Average time in minutes that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency, and data transfer times). RS-422 activity is also reported in this section.

-y Report TTY device activity:

rawch/s, canch/s, outch/s
Input character rate, input character rate processed by canon, output character rate.

rcvin/s, xmtin/s, mdmin/s
Receive, transmit and modem interrupt rates.

-c Report system calls:

scall/s system calls of all types.

sread/s, swrit/s, fork/s, exec/s
Specific system calls.

rchar/s, wchar/s Characters transferred by read and write system calls. When used with **-D**, the system calls are split into strictly local calls, remote outgoing (client) calls, and remote incoming (server) calls.

-w Report system swapping and switching activity:

swpin/s, swpot/s, bswin/s, bswot/s
Number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs).

pswch/s Process switches.

-a Report use of file access system routines: **iget/s, namei/s, dirblk/s**.

-q Report average queue length while occupied, and percentage of time occupied:

runq-sz, %runocc
Run queue of processes in memory and runnable.

swpq-sz, %swpocc
Swap queue of processes swapped out but ready to run.

-v Report status of process, i-node, file tables:

text-sz, proc-sz, inod-sz, file-sz, lock-sz
Entries/size for each table, evaluated once at sampling point.

ov Overflows that occur between sampling points for each table.

-m Report message and semaphore activities:

msg/s, sema/s Primitives per second.

-p Report paging activities:

vflt/s Address translation page faults (valid page not in memory).

pflt/s Page faults from protection errors (illegal access to page) or copy-on-writes.

pgfil/s vflt/s satisfied by page-in from file system.

rclm/s Valid pages reclaimed for free list.

-r Report unused memory pages and disk blocks:

freemem Average pages available to user processes.

freeswap Disk blocks available for process swapping.

-D Report Remote File Sharing (RFS) activity. When used in combination with **-u**, **-b** or **-c**, it causes **sar** to produce the remote file sharing version of the corresponding report. **-Du** is assumed when only **-D** is specified.

-S Report server and request queue status:

serv/lo-hi Average number of Remote File Sharing servers on the system.

request %busy Percentage of time receive descriptors are on the request queue.

request avg lgth Average number of receive descriptors waiting for service when queue is occupied.

server %avail Percentage of time there are idle servers.

server avg avail Average number of idle servers when idle ones exist

-A Report all data. Equivalent to **-udqbwcayvmprSDC**.

-C Report Remote File Sharing (RFS) buffer caching overhead:

snd-inv/s Number of invalidation messages per second sent by your machine as a server.

snd-msg/s Total outgoing RFS messages sent per second.

rcv-inv/s Number of invalidation messages received from the remote server.

rcv-msg/s Total number of incoming RFS messages received per second.

dis-bread/s Number of buffer reads that would be eligible for caching if caching were not disabled. (Indicates the penalty of running uncached.)

blk-inv/s Number of buffers removed from the client cache.

EXAMPLES

The following command reports today's CPU activity so far:

```
sar
```

The following command reports on CPU activity over a period of ten minutes and saves the data:

```
sar -o temp 60 10
```

The following command reports disk and tape activity saved from a previous *sar* (like that shown above) in which data was saved:

```
sar -d -f temp
```

FILES

/usr/adm/sa/sa*dd* daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sag(1G), sar(1M).
S/Series CTIX Administrator's Guide.

NAME

sar: sa1, sa2, sadc - system activity report package

SYNOPSIS

/usr/lib/sa/sadc [t n] [ofile]

/usr/lib/sa/sa1 [t n]

/usr/lib/sa/sa2 [**-ubdycwaqvmprDSAC**] [**-s** time] [**-e** time] [**-i** sec]

DESCRIPTION

System activity data can be accessed at the special request of a user [see *sar*(1)] and automatically on a routine basis as described here. The operating system contains a number of counters that are incremented as various system actions occur. These include counters for CPU utilization, buffer usage, disk and tape I/O activity, TTY device activity, switching and system-call activity, file-access, queue activity, interprocess communications, paging and Remote File Sharing.

The *sadc* and shell procedures, *sa1* and *sa2*, are used to sample, save, and process this data.

The data collector, *sadc*, samples system data *n* times, with an interval of *t* seconds between samples and writes in binary format to *ofile* or to standard output. If *t* and *n* are omitted, a special record is written. The *sadc* facility is used at system boot time, when booting to a multiuser state, to mark the time at which the counters restart from zero. The **/etc/init.d/perf** script checks for the presence of **/etc/rcopts/SAR**; if the startup script finds a file by that name, it uses the following command entry to write the restart mark to the daily data:

```
/bin/su - sys -c "/usr/lib/sa/sadc /usr/adm/sa/sa'date +%d' "
```

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file **/usr/adm/sa/sa***dd*, where *dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted. The following entries in **/usr/spool/cron/crontabs/sys** [see *cron*(1M)] produce records every 20 minutes during working hours and hourly otherwise:

```
0 * * * 0,6 /usr/lib/sa/sa1
20,40 8-17 * * 1-5 /usr/lib/sa/sa1
```

The shell script *sa2*, a variant of *sar*(1), writes a daily report in file **/usr/adm/sa/sar***dd*. The options are explained in *sar*(1). The following **/usr/spool/cron/crontabs/sys** entry reports important activities hourly during the working day:

```
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A &
```

The structure of the binary daily data file follows:

```
struct sa {
        struct sysinfo si;              /* see /usr/include/sys/sysinfo.h */
        struct minfo mi;                /* defined in sys/sysinfo.h */
        struct dinfo di;                /* RFS info defined in sys/sysinfo.h */
        struct rcinfo rc;               /* Client cache info defined in sys/sysinfo.h */
        struct bpbinfo bi;              /* Coprocessor info defined in sys/sysinfo.h */
        int bpb_utilize                 /* Coprocessor utilize flag */
        int minserve, maxserve;         /* RFS server low and high water marks */
        int  szinode;                   /* current size of inode table */
        int  szfile;                    /* current size of file table */
        int sztext;                     /* current size of text table */
        int  szproc;                    /* current size of proc table */
        int  szlckf;                    /* current size of file record header table */
        int  szlckr;                    /* current size of file record lock table */
        int  mszinode;                  /* size of inode table */
        int  mszfile;                   /* size of file table */
        int msztext;                    /* size of text table */
        int  mszproc;                   /* size of proc table */
        int  mszlckf;                   /* maximum size of file record header table */
        int  mszlckr;                   /* maximum size of file record lock table */
        long  inodeovf;                 /* cumulative overflows of inode table */
        long  fileovf;                  /* cumulative overflows of file table */
        long textovf;                   /* cumulative overflows of text table */
        long  procovf;                  /* cumulative overflows of proc table */
        time_t  ts;                     /* time stamp, seconds */
        int apstate;                    /* ignored */
        long  devio[NDEVS][4];          /* device unit information */
#define IO_OPS    0                   /* cumulative I/O requests */
#define IO_BCNT   1                   /* cumulative blocks transferred */
#define IO_ACT    2                   /* cumulative drive busy time in ticks */
#define IO_RESP   3                   /* cumulative I/O resp time in ticks */
};
```

FILES

/usr/adm/sa/sa*dd*	daily data file
/usr/adm/sa/sar*dd*	daily report file
/tmp/sa.adrfl	address file

SEE ALSO

cron(1M), sag(1G), sar(1), timex(1).
S/Series CTIX Administrator's Guide.

NAME

sccsdiff - compare two versions of an SCCS file

SYNOPSIS

sccsdiff **-r**SID1 **-r**SID2 [**-p**] [**-s***n*] files

DESCRIPTION

The *sccsdiff* command compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

-r*SID?*	*SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given.
-p	pipe output for each file through *pr*(1).
-s*n*	*n* is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

FILES

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), get(1), help(1), pr(1).

DIAGNOSTICS

"*file*: No differences" If the two versions are the same.
Use *help*(1) for explanations.

NAME

script - make typescript of terminal session

SYNOPSIS

script [**-a**] [**-q**] [**-S** shell] [file]

DESCRIPTION

script makes a typescript of your interaction with the system. *script* forks a shell with standard input and output diverted to pipes. Input to *script* is written to the shell's input pipe; *script* writes the shell's output pipe; and a typescript of both is written to *file*. The default for *file* is *typescript*. *File* begins and ends with time stamps for the session. *script* terminates with an error if *file* already exists.

To terminate *script*, terminate the shell or type Control-D. A Control-D to *script* terminates the shell and all programs run from the shell by closing the pipes.

The run file for the shell is taken from the **SHELL** environment variable, set by *login*(1M). If **SHELL** is not set, **/bin/sh** is used.

Here are the options:

-q Quiet operation. *script's* opening and closing messages are suppressed, as are the time stamps at the beginning and end of *file*.

-S *shell* Use *shell* as the name of the shell run file.

-a If *file* already exists, append typescript to it.

WARNINGS

script's limitations result from its use of pipes:

There is no way to send an end-of-file to the shell without terminating *script*.

Programs that use the standard input to examine and control the user's terminal will have problems or not work at all. Examples are *stty*(1), *tset*(1), *tty*(1), *ex*(1), and *vi*(1).

When the user interrupts a printing process, *script* attempts to flush the output backed up in the pipe for better response. Usually the next prompt also gets flushed.

NAME

scsimap - set mappings for SCSI devices

SYNOPSIS

scsimap [**-s** system_file] **-e** [**-d**]

scsimap [**-s** system_file] **-u** [**-d**]

scsimap [**-s** system_file] **-d**

DESCRIPTION

The *scsimap* command interrogates and sets the logical-to-physical mappings for all devices on the SCSI bus or busses. The logical SCSI device mapping determines how a physical tape or disk is mapped to an entry in the **/dev** directory.

Command line options are interpreted as follows:

-s *system_file*
: Use the file specified by *system_file*, rather than the default **/etc/system**, as the system file.

-e
: Examine the system file and check its format for consistency. The mappings that would result from running *scsimap* with the **-u** option are displayed.

-d
: Display the current kernel SCSI mappings.

-u
: Update the current kernel SCSI mappings.

System File Format

The system file consists of a number of sections, each preceded by a section header [see *system*(4)]. The *scsimap* command reads the data in the !SCSIMAP section of the system file. The format of this data consists of several lines, each line specifying a logical-to-physical mapping. The format of a line follows:

logical_device **bus=***bus* **target=***target* **lun=***logical_unit_number* **options**

logical_device
: Either **tape-***tapedrive* or **disk-***diskdrive*, where *tapedrive* is a drive number **d0** to **d7** and *diskdrive* is a drive number **c0d0** to **c0df**. The drive number corresponds to the logical device number: for example, if *tapedrive* is **4**, the device **/dev/rmt/c***n***d4** is mapped.

bus
: The SCSI bus number: **0** for onboard SCSI, **1** to **4** for the I/O slot number of the SCSI combo board.

target
: The target address of the device, **0** to **6**. (7 is the host ID.)

logical_unit_number The logical unit number of the device, **0** to **3** (usually 0).

options **parity**, **reselect**, and **halfinch**.

The **parity** option indicates that the target generates parity errors when they occur; **reselect** indicates that the target can handle disconnect/reselect protocol; **halfinch** indicates that the target is a half-inch tape drive.

EXAMPLE

The following example shows the **!SCSIMAP** section of the **/etc/system** file for a system with five SCSI disks, a SCSI quarter-inch cartridge (QIC) tape, and two SCSI half-inch tapes.

Note that the bus number is system expansion slot-dependent. If the board is in slot number 2, the bus number is 2.

Once you select the drive target number, remember to set the physical drive switches or jumpers to that target number.

```
!SCSIMAP
disk-c0d0  bus=0  target=6  lun=0  parityreselect
disk-c0d1  bus=0  target=5  lun=0  parityreselect
tape-d0    bus=0  target=1  lun=0  parityreselect
tape-d1    bus=0  target=2  lun=0  parityreselect halfinch
disk-c0d2  bus=1  target=6  lun=0  parityreselect
disk-c0d3  bus=1  target=5  lun=0  parityreselect
disk-c0d4  bus=1  target=4  lun=0  parityreselect
tape-d2    bus=2  target=0  lun=0  parityreselect halfinch
```

Disks 0 and 1 are connected to the onboard SCSI bus, with target IDs of 6 and 5, respectively. The SCSI QIC tape is connected to the onboard SCSI bus, with a target ID of 1. The first SCSI half-inch tape is connected to the onboard SCSI bus, with a target ID of 2. Disks 2, 3, and 4 are connected to the SCSI RS-232 board in the first slot, with target IDs of 6, 5, and 4, repectively. The second SCSI half-inch tape is connected to the SCSI RS-232 board in the second slot, with a target ID of 0. Note that the disk controller number is always **c0** and that the range for disk drives is **0** through **f**; the range for tape drives is **0** through **7**.

DIAGNOSTICS

If *scsimap* encounters errors in accessing, or in the entries of the system_file, it prints error messages. Some of the possible errors follow:

Unable to access system_file.

Unable to write to kernel memory.

Ill-formed SCSIMAP section entry lines.

Multiple targets on a bus.

More than seven targets on one bus.

Too many disk drives or tape drives in the map.

FILES

/dev/scsi
/etc/system
/usr/sys/cf/*dfile*

SEE ALSO

system(4), scsi(7).
S/Series CTIX Administrator's Guide.

WARNINGS

If *scsimap* encounters errors in the system_file being used to update the kernel SCSI mapping, or if it cannot successfully read from or write to kernel memory, it prints an error message and does not update the map.

On an S/80, on bus 0, target 0 is reserved for the SCSI LAN Board; avoid using this target number if you plan to use the SCSI LAN board.

NAME

sdb - symbolic debugger

SYNOPSIS

sdb [**-w**] [**-W**] [objfil [corfil [directory-list]]]

DESCRIPTION

The *sdb* command calls a symbolic debugger that can be used with C programs. It can be used to examine their object files and core files and to provide a controlled environment for their execution.

Objfil is an executable program file that has been compiled with the **-g** (debug) option. If *objfil* has not been compiled with the **-g** option, the symbolic capabilities of *sdb* are limited, but the file can still be examined and the program debugged. 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**. The core file need not be present. A **-** in place of *corfil* forces *sdb* to ignore any core image file. The colon-separated list of directories (*directory-list*) is used to locate the source files used to build *objfil*.

It is useful to know that at any time there is a *current line* and *current file*. If *corfil* exists, they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in *main()*. The current line and file can be changed by using the source file examination commands.

Initially *sdb* has an asterisk character (*) prompt, which indicates that *sdb* is ready for the user to enter the first command. If the **S**, **s**, **I**, or **i** command is used, the prompt corresponds to the command letter (for example, **S** when the **S** command is used).

By default, warnings are provided if the source files used in producing *objfil* cannot be found, or are newer than *objfil*. This checking feature and the accompanying warnings may be disabled by the use of the **-W** flag.

Names of variables are written just as they are in C. *sdb* does not truncate names. Variables local to a procedure may be accessed using the form *procedure*:*variable*. If no procedure name is given, the procedure containing the current line is used by default.

It is also possible to refer to structure members as *variable*.*member*, pointers to structure members as *variable*->*member* and array elements as *variable*[*number*]. Pointers may be dereferenced by using the form *pointer*[0]. Combinations of these forms may also be used. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last

structure referenced by *sdb*. An unqualified structure variable may also be used with various commands. Generally, *sdb* will interpret a structure as a set of variables. Thus, *sdb* will display the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value *sdb* displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as *variable [number][number]...*, or as *variable [number,number,...]*. In place of *number*, the form *number;number* may be used to indicate a range of values, * may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, *sdb* displays all the values of an array or of the section of an array if trailing subscripts are omitted. It displays only the address of the array itself or of the section specified by the user if subscripts are omitted.

A particular instance of a variable on the stack may be referenced by using the form *procedure:variable,number*. All the variations mentioned in naming variables may be used. *Number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C may be used, so that addresses may be input in decimal, octal or hexadecimal.

Line numbers in the source program are referred to as *file-name:number* or *procedure:number*. In either case the number is relative to the beginning of the file. If no procedure or file name is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under *sdb*, all addresses refer to the executing program; otherwise they refer to *objfil* or *corfil*. An initial argument of **-w** permits overwriting locations in *objfil*.

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<=address<e1

file address=address+f1-b1

otherwise

b2<=address<e2

file address=address+f2-b2,

otherwise, the requested *address* is not legal. In some cases (for example, for programs with separated I and D space) the two segments for a file may overlap.

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.

In order for *sdb* to be used on large files, all appropriate values are kept as signed 32-bit integers.

Commands

The commands for examining data in the program are:

t Print a stack trace of the terminated or halted program.

T Print the top line of the stack trace.

variable/*clm*

Print the value of *variable* according to length *l* and format *m*. A numeric count *c* indicates that a region of memory, beginning at the address implied by *variable*, is to be displayed. The length specifiers are:

b one byte

h two bytes (half word)

l four bytes (long word)

Legal values for *m* are:

c character

d decimal

u decimal, unsigned

o octal

x	hexadecimal
f	32-bit single precision floating point
g	64-bit double precision floating point
s	Assume *variable* is a string pointer and print characters starting at the address pointed to by the variable.
a	Print characters starting at the variable's address. This format may not be used with register variables.
p	pointer to procedure
i	disassemble machine-language instruction with addresses printed numerically and symbolically.
I	disassemble machine-language instruction with addresses just printed numerically.

Length specifiers are only effective with the **c**, **d**, **u**, **o** and **x** formats. Any of the specifiers, *c*, *l*, and *m*, may be omitted. If all are omitted, *sdb* chooses a length and a format suitable for the variable's type as declared in the program. If *m* is specified, then this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier *c* tells *sdb* to display that many units of memory, beginning at the address of *variable*. The number of bytes in one such unit of memory is determined by the length specifier *l*, or if no length is given, by the size associated with the *variable*. If a count specifier is used for the **s** or **a** command, then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command *./*.

The *sh*(1) metacharacters * and ? may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, variables local to the current procedure and global variables are matched; if a procedure name is specified then only variables local to that procedure are matched. To match only global variables, the form *:pattern* is used.

linenumber?*lm*
variable:?*lm*

Print the value at the address from **a.out** or I space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is 'i'.

variable=*lm*
linenumber=*lm*
number=*lm*

Print the address of *variable* or *linenumber*, or the value of *number*, in the format specified by *lm*. If no format is given, then **lx** is used. The last variant of this command provides a convenient way to convert between decimal, octal and hexadecimal.

variable!*value*

Set *variable* to the given *value*. The value may be a number, a character constant or a variable. The value must be well defined; expressions which produce more than one value, such as structures, are not allowed. Character constants are denoted '*character*. Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type double. Registers are viewed as integers. The *variable* may be an expression which indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type *int*. C conventions are used in any type conversions necessary to perform the indicated assignment.

f Print the 68881 floating-point registers.

x Print the machine registers and the current machine-language instruction.

X Print the current machine-language instruction.

The commands for examining source files are:

e *procedure*
e *file-name*
e *directory/*
e *directory file-name*

The first two forms set the current file to the file containing *procedure* or to *file-name*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The default is the current working directory. The latter two forms change the value of *directory*. If no procedure, file name, or directory is given, the current procedure name and file name are reported.

/regular expression/

Search forward from the current line for a line containing a string matching *regular expression* as in *ed*(1). The trailing / may be deleted.

?regular expression?
Search backward from the current line for a line containing a string matching *regular expression* as in *ed*(1). The trailing ? may be deleted.

p Print the current line.

z Print the current line followed by the next 9 lines. Set the current line to the last line printed.

w Window. Print the 10 lines around the current line.

number
Set the current line to the given line number. Print the new current line.

count+
Advance the current line by *count* lines. Print the new current line.

count−
Retreat the current line by *count* lines. Print the new current line.

The commands for controlling the execution of the source program are:

count **r** *args*
count **R**
Run the program with the given arguments. The **r** command with no arguments reuses the previous arguments to the program while the **R** command runs the program with no arguments. An argument beginning with < or > causes redirection for the standard input or output, respectively. If *count* is given, it specifies the number of breakpoints to be ignored.

linenumber **c** *count*
linenumber C *count*
Continue after a breakpoint or interrupt. If *count* is given, the program will stop when *count* breakpoints have been encountered. The signal which caused the program to stop is reactivated with the C command and ignored with the **c** command. If a line number is specified, a temporary breakpoint is placed at the line and execution is continued. This temporary breakpoint is deleted when the command finishes.

linenumber **g** *count*
Continue after a breakpoint with execution resumed at the given line. If *count* is given, it specifies the number of breakpoints to be ignored.

s *count*
S *count*

Single step the program through *count* lines. If no count is given then the program is run for one line. **S** is equivalent to **s** except it steps through procedure calls.

i
I Single step by one machine-language instruction. The signal which caused the program to stop is reactivated with the **I** command and ignored with the **i** command.

variable$**m** *count*
address:**m** *count*

Single step (as with **s**) until the specified location is modified with a new value. If *count* is omitted, it is effectively infinity. *Variable* must be accessible from the current procedure. Since this command is done by software, it can be very slow.

level **v**

Toggle verbose mode, for use when single stepping with **S**, **s** or **m**. If *level* is omitted, then just the current source file and/or subroutine name is printed when either changes. If *level* is 1 or greater, each C source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A **v** turns verbose mode off if it is on for any level.

k Kill the program being debugged.

procedure(arg1,arg2,...)
procedure(arg1,arg2,...)/*m*

Execute the named procedure with the given arguments. Arguments can be integer, character or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format *m*. If no format is given, it defaults to **d**. This facility is only available if the program was loaded with the -g option.

linenumber **b** *commands*

Set a breakpoint at the given line. If a procedure name without a line number is given (for example, "proc:"), a breakpoint is placed at the first line in the procedure even if it was not compiled with the **-g** option. If no *linenumber* is given, a breakpoint is placed at the current line. If no *commands* are given, execution stops just before the breakpoint and control is returned to *sdb*. Otherwise the *commands* are executed when the breakpoint is encountered and execution continues. Multiple

commands are specified by separating them with semicolons. If **k** is used as a command to execute at a breakpoint, control returns to *sdb*, instead of continuing execution.

B Print a list of the currently active breakpoints.

linenumber **d**
Delete a breakpoint at the given line. If no *linenumber* is given then the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from the standard input. If the line begins with a **y** or **d** then the breakpoint is deleted.

D Delete all breakpoints.

l Print the last executed line.

linenumber **a**
Announce. If *linenumber* is of the form *proc*:*number*, the command effectively does a *linenumber* **b l**. If *linenumber* is of the form *proc*:, the command effectively does a *proc*: **b T**.

Miscellaneous commands:

!*command*
The command is interpreted by *sh*(1).

new-line
Perform the previous command again.

end-of-file character
Scroll. Print the next 10 lines of instructions, source or data depending on which was printed last. The end-of-file character is usually control-D.

< *filename*
Read commands from *filename* until the end of file is reached, and then continue to accept commands from standard input. When *sdb* is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; < may not appear as a command in a file.

M Print the address maps.

M [?/] [*] *b e f*
Record new values for the address map. The arguments **?** and / specify the text and data maps, respectively. The first segment (*b1*, *e1*, *f1*) is changed unless * is specified, in which case the second segment (*b2*, *e2*, *f2*) of the mapping is changed. If fewer than three values are given, the remaining map parameters are left unchanged.

" *string*
Print the given string. The C escape sequences of the form *character* are recognized, where *character* is a nonnumeric character.

q Exit the debugger.

The following commands also exist and are intended only for debugging the debugger:

V Print the version number.
Q Print a list of procedures and files being debugged.
Y Toggle debug output.

sdb may be instructed to monitor a given memory location and stop the program when the value at that location changes in any given way. For example:

```
> if x <= 123
```

The above example instructs *sdb* to monitor the value at location x. When the user gives the command to continue (c), *sdb* checks the value of x at every source line executed and stops the program if the given condition becomes true. Note that use of this construct slows the real-time execution of a program.

The syntax of the *if* command is as follows:

if Shows a list of the current data breakpoints; assigns a number to each.

if *var* Monitors the value of *var* and stops the program if the value changes. A variable name may be used for *var*, as well as a constant address. Comparisons are done as either 4-byte signed or 4-byte unsigned, depending on the data type. To perform a 1-byte or 2-byte comparison, an optional length value may accompany *var*. An example of a 2-byte comparison is

```
if x,2 = 0xff
```

if *var rel value*
Compares the value of *var* to the constant given and stops the program if the condition is true. The values of *rel* may be =, ==, <, <=, >, >=, or !=.

off *n* Disables or turns off a data breakpoint without removing it from the list.

on *n* Enables a breakpoint that was turned off.

out *n* Removes a breakpoint from the list.

Conditional breakpoints are used in a manner similar to data breakpoints, except that the user specifies a place in the program at which *sdb* should stop to check the data values. For example,

mysub:99 b if xyz = 123

The above example instructs *sdb* to check the value of xyz every time the program arrives at line 99 of subroutine *mysub*. If the condition is true, then execution stops there, as with a normal breakpoint. This type of breakpoint does not monitor the value xyz at every line of code, as the data breakpoint does.

FILES

a.out
core

SEE ALSO

cc(1), sh(1), a.out(4), core(4), syms(4).

WARNINGS

When *sdb* prints the value of an external variable for which there is no debugging information, a warning is printed before the value. The size is assumed to be **int** (integer).

Data which are stored in text sections are indistinguishable from functions.

Line number information in optimized functions is unreliable, and some information may be missing.

BUGS

If a procedure is called when the program is *not* stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

When setting a breakpoint at a procedure, *sdb* will inconsistently produce the incorrect line number. Recompiling the source program will correct this problem.

NAME

sdiff - side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

sdiff uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
x       |       y
a               a
b       <
c       <
d               d
        >       c
```

The following options exist:

-w *n* Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.

-l Only print the left side of any lines that are identical.

-s Do not print identical lines.

-o *output* Use the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a **%** and waits for one of the following user-typed commands:

- **l** append the left column to the output file
- **r** append the right column to the output file
- **s** turn on silent mode; do not print identical lines
- **v** turn off silent mode
- **e l** call the editor with the left column

e r	call the editor with the right column
e b	call the editor with the concatenation of left and right
e	call the editor with a zero length file
q	exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1).

NAME

sed - stream editor

SYNOPSIS

sed [**-n**] [**-e** script] [**-f** sfile] [files]

DESCRIPTION

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** options, 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.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, that is, a */regular expression/* in the style of *ed*(1) modified thus:

In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

The escape sequence **\n** matches a new-line *embedded* in the pattern space.

A period . matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second

address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with \ to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) **a**
text Append. Place *text* on the output before reading the next input line.

(2) **b** *label* Branch to the **:** command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) **c**
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) **d** Delete the pattern space. Start the next cycle.

(2) **D** Delete the initial segment of the pattern space through the first new-line. Start the next cycle.

(2) **g** Replace the contents of the pattern space by the contents of the hold space.

(2) **G** Append the contents of the hold space to the pattern space.

(2) **h** Replace the contents of the hold space by the contents of the pattern space.

(2) **H** Append the contents of the pattern space to the hold space.

(1) **i**
text Insert. Place *text* on the standard output.

(2) **l** List the pattern space on the standard output in an unambiguous form. Non-printable characters are displayed in octal notation ASCII and long lines are folded.

(2) **n** Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) **N** Append the next line of input to the pattern space with an embedded new-line. (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 new-line to the standard output.

(1) **q** Quit. Branch to the end of the script. Do not start a new cycle.

(2) **r** *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) **s**/*regular expression*/*replacement*/*flags*

Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed*(1). *Flags* is zero or more of:

n n= 1 - 512. Substitute for just the n th occurrence of the *regular expression.*

g Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Print the pattern space if a replacement was made.

w *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) **t** *label* Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent 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.

(0) # If a # appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the # is an 'n', then the default output will be suppressed. The rest of the line after #n is also ignored. A script file must contain at least one non-comment line.

SEE ALSO

awk(1), ed(1), grep(1).
Programmer's Guide: CTIX Supplement.

NAME

sendmail - mail routing program

SYNOPSIS

/usr/lib/sendmail [flags] [address ...]

newaliases

mailq [**-v**]

DESCRIPTION

sendmail sends a message to one or more *recipients*, routing the message over whatever networks are necessary. *sendmail* does internetwork forwarding as necessary to deliver the message to the correct place.

sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; *sendmail* is used only to deliver pre-formatted messages.

With no flags, *sendmail* reads its standard input up to an end-of-file or a line consisting only of a single dot and sends a copy of the message found there to all of the addresses listed. It determines the network(s) to use based on the syntax and contents of the addresses.

sendmail supports the Simple Mail Transfer Protocol (SMTP) as documented in RFC 821.

Local addresses are looked up in a file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in any alias expansions, for example, if 'john' sends to 'group', and 'group' includes 'john' in the expansion, then the letter will not be delivered to 'john'.

Flags are:

-ba Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the "From:" and "Sender:" fields are examined for the name of the sender.

-bd Run as a daemon. This requires TCP/IP. *sendmail* will fork and run in background listening on the SMTP socket [see *services*(4)]. This is normally run from **/etc/rcopts/NETWORK**.

-bi Initialize the alias database.

-bm Deliver mail in the usual way (default).

-bp Print a listing of the queue.

-bs Use the SMTP protocol as described in RFC 821 on standard input and output. This flag implies all the operations of the **-ba** flag that are compatible with SMTP.

-bt Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.

-bv Verify names only - do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.

-bz Create the configuration freeze file.

-C*file* Use alternate configuration file. *sendmail* refuses to run as root if an alternate configuration file is specified. The frozen configuration file is bypassed.

-d*X* Set debugging value to *X*.

-F*fullname* Set the full name of the sender.

-f*name* Sets the name of the ''from'' person (that is, the sender of the mail). **-f** can only be used by ''trusted'' users (normally *root, daemon*, and *network)* or if the person you are trying to become is the same as the person you are.

-h*N* Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop. If not specified, ''Received:'' lines in the message are counted.

-n Don't do aliasing.

-o*x value* Set option *x* to the specified *value*. Options are described below.

-q[*time*] Processed saved messages in the queue at given intervals. If *time* is omitted, process the queue once. *Time* is given as a tagged number, with 's' being seconds, 'm' being minutes, 'h' being hours, 'd' being days, and 'w' being weeks. For example, ''-q1h30m'' or ''-q90m'' would both set the timeout to one hour thirty minutes. If *time* is specified, *sendmail* will run in background. This option can be used safely with **-bd**.

-t Read message for recipients. To:, Cc:, and Bcc: lines will be scanned for recipient addresses. The Bcc: line will be deleted before transmission. Any addresses in the argument list will be suppressed, that is, they will *not* receive copies even if listed in the message header.

-v Go into verbose mode. Alias expansions will be announced, etc.

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **-o** flag or in the configuration file, **/usr/lib/sendmail.cf**. The options are:

A*file* Use alternate alias file.

c On mailers that are considered "expensive" to connect to, don't initiate immediate connection. This requires queueing.

d*x* Set the delivery mode to *x*. Delivery modes are 'i' for interactive (synchronous) delivery, 'b' for background (asynchronous) delivery, and 'q' for queue only - that is, actual delivery is done the next time the queue is run.

D Try to automatically rebuild the alias database if necessary.

e*x* Set error processing to mode *x*. Valid modes are 'm' to mail back the error message, 'w' to "write" back the error message (or mail it back if the sender is not logged in), 'p' to print the errors on the terminal (default), and 'q' to throw away error messages (only exit status is returned). If the text of the message is not mailed back by modes 'm' or 'w' and if the sender is local to this machine, a copy of the message is appended to the file "dead.letter" in the sender's home directory.

F*mode* The mode to use when creating temporary files.

f Save UNIX-style From lines at the front of messages.

g*N* The default group id to use when calling mailers.

H*file* The SMTP help file.

i	Do not take dots on a line by themselves as a message terminator.
L*n*	The log level.
m	Send to "me" (the sender) also if I am in an alias expansion.
o	If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (that is, commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.
Q*queuedir*	Select the directory in which to queue messages.
r*timeout*	The timeout on reads; if none is set, *sendmail* will wait forever for a mailer. This option violates the word (if not the intent) of the SMTP specification, show the timeout should probably be fairly large.
S*file*	Save statistics in the named file.
s	Always instantiate the queue file, even under circumstances where it is not strictly necessary. This provides safety against system crashes during delivery.
T*time*	Set the timeout on undelivered messages in the queue to the specified time. After delivery has failed (for example, because of a host being down) for this amount of time, failed messages will be returned to the sender. The default is three days.
t*stz,dtz*	Set the name of the time zone.
u*N*	Set the default user id for mailers.

In aliases, the first character of a name may be a vertical bar to cause interpretation of the rest of the name as a command to pipe the mail to. It may be necessary to quote the name to keep *sendmail* from suppressing the blanks from between arguments. For example, a common alias is:

```
msgs: "|/usr/local/msgs -s"
```

Aliases may also have the syntax ":include:*filename*" to ask *sendmail* to read the named file for a list of recipients. For example, an alias such as:

poets: ":include:/usr/local/poets.list"

would read **/usr/local/poets.list** for the list of addresses making up the group.

sendmail returns an exit status describing what it did. The codes are defined in **<sysexits.h>**

EX_OK	Successful completion on all addresses.
EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as "cannot fork".
EX_NOHOST	Host name not recognized.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

If invoked as *newaliases*, *sendmail* will rebuild the alias database. If invoked as *mailq*, *sendmail* will print the contents of the mail queue.

FILES

Except for **/usr/lib/sendmail.cf** and **/bin/rmail**, these pathnames are all specified in **/usr/lib/sendmail.cf**. Thus, these values are only approximations.

/bin/rmail	mail delivery program
/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.pag	
/usr/lib/aliases.dir	data base of alias names
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	help file
/usr/lib/sendmail.st	collected statistics
/usr/spool/mqueue/*	temp files

SEE ALSO

mail(1), newaliases(1), rmail(1), aliases(4), mailaddr(5).
RFC 819, RFC 821, RFC 822.

NAME

serstat - display serial port error statistics

SYOPNSIS

serstat

DESCRIPTION

The *serstat* command reports error status information about groups of serial tty ports. The command does not currently support IOP and RIOP ports. When first invoked, *serstat* finds the four ports with the largest number of total errors logged and displays the logged errors.

The command then runs in "automatic" mode, in which it scans all serial ports for any change of status As port status changes, *serstat* updates the display to ensure that the four ports with the largest number of errors logged are displayed at all times. Ports with fewer errors logged are replaced as other ports with more errors logged are displayed. A message at the bottom of the screen indicates which port has most recently changed.

The *serstat* program can also be run in "scan" mode and "continuous" mode. In scan mode, *serstat* scans sequential groups of ports every three seconds and displays the errors. In continuous, *serstat* continues to scan and update the currently-displayed ports only.

To exit *serstat*, generate a keyboard interrupt.

Once *serstat* is running, use any of the following one-character commands:

r Redraw the screen. No mode change.

a Redraw the screen. Start automatic mode.

m Redraw the screen with ports having the most errors. Start automatic mode.

s Redraw the screen. Start scan mode.

c Redraw the screen. Start continuous mode.

The program displays data from the following status structure maintained by the serial driver in the kernel:

```
struct sererrstat {
        uint    se_ttyhog;      /* tty input hog status achieved (ttin) */
        uint    se_iflushed;    /* hogs input queues discarded (ttin) */
        uint    se_idropped;    /* input char(s) dropped (ttin, serrint) */
        uint    se_norbuf;      /* no receive buffer available (serrint) */
        uint    se_othrottle;   /* output throttled, low clists (T_HIWATER) */
        uint    se_oflushed;    /* hogs output queue discarded (ttxput) */
```

```
        uint    se_odropped;    /* output char(s) dropped (serxsend, sersend) */
        uint    se_notbuf;      /* no transmit buffer available (ttout) */
        uint    se_rxorun;      /* receiver overrun (serrint) */
        uint    se_exstat;      /* external status change (sertint) */
        uint    se_pe;          /* parity errors (serrint) */
        uint    se_frame;       /* CRC/framing error (serrint) */
};
```

All fields are incremented once per event occurrence except **se_idropped** and **se_odropped**. These two fields try to keep track of the number of characters dropped for that particular error event instead of the number of times that error event occurred. Note that 256 characters or more can be lost when the input queue is flushed, but the only record of this event is a single increment to the **se_iflushed** field.

The field **se_exstat** counts the number of external status changes occurring on a port. A break condition or change in the Carrier Detect or Clear To Send lines increments this number. These are not normally error conditions, but may be of interest.

SEE ALSO

termio(7).

NOTE

This utility is intended for diagnostic use by qualified system administrators; it is not a basic user command.

NAME

setmnt - establish mount table

SYNOPSIS

/etc/setmnt

DESCRIPTION

setmnt creates the **/etc/mnttab** table which is needed for both the *mount*(1M) and *umount* commands. *setmnt* reads standard input and creates a *mnttab* entry for each line. Input lines have the format:

filesys node

where *filesys* is the name of the file system's *special file* (for example, **/dev/dsk/c?d?s?**) and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the mount table entry.

FILES

/etc/mnttab

SEE ALSO

mount(1M).

BUGS

Problems may occur if *filesys* or *node* are longer than 32 characters.

setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.

NAME

setuname - set name of system

SYNOPSIS

/etc/setuname [-s sysname] [**-n** nodename] [**-r** release] [**-v** *version*]

DESCRIPTION

setuname sets the values reported by *uname*. Options set the same things that they report in *uname*. *Sysname*, *nodename*, and *release* are truncated to eight characters. Note that **setuname -n** *nodename* resets the nodename component of the Internet hostname [see *hostname*(1)].

See *hostname*(1) for information about how the nodename as returned by *uname*(1) is set when the system is rebooted.

Only the superuser can execute *setuname* successfully.

SEE ALSO

rc2(1M), uname(1), uname(2).

NAME

sh, rsh - shell, the standard/restricted command programming language

SYNOPSIS

sh [**-acefhiknrstuvx**] [args]

rsh [**-acefhiknrstuvx**] [args]

DESCRIPTION

The *sh* interpreter provides a command programming language that executes commands read from a terminal or a file. The *rsh* interpreter is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See "Invocation" for the meaning of arguments to the shell.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *, @, #, ?, -, $, and !.

Commands

A *simple-command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 [see *exec*(2)]. The *value* of a *simple-command* is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally [see *signal*(2) for a list of status values].

A *pipeline* is a sequence of one or more *commands* separated by |. The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by ;, &, &&, or | |, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and | |. The symbols && and | | also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (the shell does *not* wait for that pipeline to finish). The symbol && (| |) causes the *list* following it to be executed only if the

preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a *simple-command* or one of the following. Unless otherwise stated, the value returned by a command is that of the last *simple-command* executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, the **for** command executes the **do** *list* once for each positional parameter that is set (see "Parameter Substitution"). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ... *list* ;;] ... **esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see "File Name Generation") except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, the **if** command returns a zero exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, the **while** command returns a zero exit status; **until** can be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a sub-shell.

{*list*;}

list is executed in the current (that is, parent) shell.

name () {*list*;}

Define a function referred to by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see "Execution").

The following words are recognized only as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Command Substitution

The shell reads commands from the string between two grave accents (`` ` ` ``) and the standard output from these commands can be used as all or part of a word. Trailing new-lines from the standard output are removed.

No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes can be used to escape a grave accent (`) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes (" ...` ...` ... "), a backslash used to escape a double quote (\") is removed; otherwise, it is left intact.

If a backslash is used to escape a new-line character (**new-line**), both the backslash and the new-line are removed (see "Quoting"). In addition, backslashes used to escape dollar signs (**$**) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \, `, ", **new-line**, and **$** are left intact when the command string is read.

Parameter Substitution

The character **$** is used to introduce substitutable *parameters*. There are two types of parameters: positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters can be assigned values by **set**. Keyword parameters (also known as variables) can be assigned values as follows:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. A function and a variable must not use the same *name*.

${*parameter*}

The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If

parameter is * or @, all positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter*:-*word*}
If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters cannot be assigned to in this way.

${*parameter*:?*word*}
If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:+*word*}
If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string; in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, the shell checks only whether *parameter* is set or not.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.

- Flags supplied to the shell on invocation or by the **set** command.

? The decimal value returned by the last synchronously executed command.

$ The process number of this shell.

! The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.

PATH The search path for commands (see "Execution"). The user may not change **PATH** if executing under *rsh*.

CDPATH
The search path for the *cd* command.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

MAILCHECK
How often (in seconds) the shell checks for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell checks before each prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that is printed when the modification time changes. The default message is *you have mail*.

PS1 Primary prompt string, by default "$ ".

PS2 Secondary prompt string, by default "> ".

IFS Internal field separators, normally **space**, **tab**, and **new-line**.

SHACCT
If this parameter is set to the name of a file writable by the user, the shell writes an accounting record in the file for each shell procedure executed. Accounting routines such as *acctcom*(1) and *acctcms*(1M) can be used to analyze the data collected.

SHELL When the shell is invoked, it scans the environment (see "Environment") for this name. If it is found and **rsh** is the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS**. **HOME** and **MAIL** are set by *login*(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Input/Output

A command's input and output may be redirected using a special notation interpreted by the shell. The following can appear anywhere in a *simple-command* or can precede or follow a *command* and are *not* passed on as arguments to the invoked command. Note that parameter and command substitution occurs before *word* or *digit* is used.

<word Use file *word* as standard input (file descriptor 0).

>word Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

>>word Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[-]word After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file. If, however, **-** is appended to **<<**:

1) leading tabs are stripped from *word* before the shell input is read (but after parameter and command substitution is done on *word*),

2) leading tabs are stripped from the shell input as it is read and before each line is compared with *word*, and

3) shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file.

If any character of *word* is quoted (see "Quoting"), no additional processing is done to the shell input. If no characters of *word* are quoted:

1) parameter and command substitution occurs,

2) (escaped) **\new-line** is ignored, and

3) \ must be used to quote the characters \, $, and `.

The resulting document becomes the standard input.

<&digit Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using **>&digit**.

<&- The standard input is closed. Similarly for the standard output using **>&-**.

If any of the above is preceded by a digit, the file descriptor to be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

... 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (*xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

Using the terminology introduced on the first page, under "Commands," if a *command* is composed of several *simple commands*, redirection is evaluated for the entire *command* before it is evaluated for each *simple command*. That is, the shell evaluates redirection for the entire *list*, then each *pipeline* within the *list*, then each *command* within each *pipeline*, then each *list* within each *command*.

If a command is followed by **&** the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

File Name Generation

Before a command is executed, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically

between the pair, inclusive. If the first character following the opening ``[´´ is a "!" any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > new-line space tab

A character can be *quoted* (made to stand for itself) by preceding it with a backslash (\) or inserting it between a pair of quote marks (´´ or ""). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair **\new-line** is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (´´), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote can be quoted inside a pair of double quote marks (for example, " ´ ").

Inside a pair of double quote marks (""), parameter and command substitution occurs and the shell quotes the results to avoid blank interpretation and file name generation. If **$*** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces ("**$1 $2** ..."); however, if **$@** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces ("**$1**" "**$2**" ...). \ quotes the characters \, `, ", and $. The pair **\new-line** is removed before parameter and command substitution. If a backslash precedes characters other than \, `, ", $, and new-line, the backslash is quoted by the shell.

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (that is the value of PS2) is issued.

Environment

The *environment* [see *environ*(5)] is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affect the environment unless the **export** command is used to bind the shell's parameter to the environment (see

also **set -a**). A parameter can be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* can be augmented by prefixing it with one or more assignments to parameters. Thus, the following two command lines are equivalent (as far as the execution of *cmd* is concerned):

```
TERM=450 cmd
```

and

```
(export TERM; TERM=450; cmd)
```

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following command sequence first prints **a=b c** and **c**:

```
echo a=b c
set -k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise, signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null

path name, which can appear immediately after the equal sign, between two colon delimiters anywhere in the path list, or at the end of the path list. If the command name contains a / the search path is not used; such commands are not executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be redetermined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command (described later) is executed.

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command cannot be executed by *rsh*.

echo [*arg* ...]
Echo arguments. See *echo*(1) for usage and description.

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]
The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are *not* exported.

getopts Use in shell scripts to support command syntax standards [see *intro*(1)]; it parses positional parameters and checks for legal options. See *getopts* (1) for usage and description.

hash [**-r**] [*name* ...]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this is done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* are incremented when the recalculation is done.

newgrp [*arg* ...]
Equivalent to **exec newgrp** *arg* .... See *newgrp*(1) for usage and description.

pwd Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]
One line is read from the standard input and, using the internal field separator, **IFS** (normally space or tab), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, and so on, with leftover words assigned to the last *name*. Lines can be continued using **\new-line**. Characters other than **new-line** can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *name*s are marked *readonly* and the values of the these *name*s cannot be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**--aefhkntuvx** [*arg* ...]]

-a Mark variables that are modified or created for export.

-e Exit immediately if a command exits with a non-zero exit status.

-f Disable file name generation.

-h Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).

-k All keyword arguments are placed in the environment for a command, not just those that precede the command name.

-n Read commands but do not execute them.

-t Exit after reading and executing one command.

-u Treat unset variables as an error when substituting.

-v Print shell input lines as they are read.

-x Print commands and their arguments as they are executed.

-- Do not change any of the flags; end the option list; for example, **set -- -** sets **$1** to -.

Using + rather than - causes these flags to be turned off. Using - by itself is equivalent to using **+xv**. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$-**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given, the values of all names are printed.

shift [*n*]

The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

test

Evaluate conditional expressions. See *test*(1) for usage and description.

times

Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...

The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]

For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [*n*]

Impose a size limit of *n* blocks on files written by the shell and its child processes (files of any size may be read). If *n* is omitted, the current limit is printed. You can lower your own ulimit, but only a super-user can raise a ulimit [see *su*(1M)].

umask [*nnn*]
The user file-creation mask is set to *nnn* [see *umask*(1)]. If *nnn* is omitted, the current value of the mask is printed.

unset [*name* ...]
For each *name*, remove the corresponding variable or function. The variables **PATH, PS1, PS2, MAILCHECK** and **IFS** cannot be unset.

wait [*n*]
Wait for your background process whose process ID is *n* and report its termination status. If *n* is omitted, all your shell's currently-active background processes are waited for and the return code will be zero.

Invocation

If the shell is invoked through *exec*(2) and the first character of argument zero is -, commands are initially read from **/etc/profile** and from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as */bin/sh*. The flags below are interpreted by the shell on invocation only. Note that unless the **-c** or **-s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

-c *string* If the **-c** flag is present commands are read from *string*.

-s If the **-s** flag is present, or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.

-i If the **-i** flag is present, or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

-r If the **-r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

rsh Only

The *rsh* command interpreter is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory [see *cd*(1)],
setting the value of **$PATH,**
specifying path or command names containing /,
redirecting output (> and >>).

The restrictions above are enforced after **.profile** is interpreted.

A restricted shell can be invoked in one of the following ways: (1) *rsh* is the file name part of the last entry in the **/etc/passwd** file [see *passwd*(4)]; (2) the environment variable **SHELL** exists and *rsh* is the file name part of its value; (3) the shell is invoked and *rsh* is the file name part of argument 0; (4) the shell is invoke with the **-r** option.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** [see *profile*(4)] has complete control over user actions by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (for example, **/usr/rbin**) that can be safely invoked by a restricted shell. Some systems also provide a restricted editor, *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see the **exit** command above).

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

acctcom(1), acctcms(1M), cd(1), csh(1), echo(1), env(1), getopts(1), intro(1), login(1), newgrp(1), pwd(1), test(1), umask(1), wait(1), dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2), profile(4).

NOTE

If the first character in an executable file is #, *csh* assumes that the file is a *csh* script. For compatibility with *csh*, it is recommended that *sh* scripts begin with a blank line.

CAVEATS

Words used for filenames in input/output redirection are not interpreted for filename generation (see "File Name Generation," above). For example, cat **file1 >a*** creates a file named **a***.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message *cannot fork, too many processes*, try using the *wait* (1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process IDs associated with your login, and to the number the system can keep track of.)

BUGS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell continues to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For *wait n*, if *n* is not an active process id, all your shell's currently active background processes are waited for and the return code is zero.

NAME

shl - shell layer manager

SYNOPSIS

shl

DESCRIPTION

The *shl* program allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as *layers*, using the commands described below.

The *current layer* is the layer that can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To block the output of a layer when it is not current, the *stty* option **loblk** can be set within the layer.

The *stty* character **swtch** (set to ^Z if NUL) is used to switch control to *shl* from a layer. The *shl* prompt, >>>, distinguishes *shl* from a layer.

A *layer* is a shell that has been bound to a virtual tty device (**/dev/sxt/???**). The virtual device can be manipulated like a real tty device by using *stty* (1) and *ioctl* (2). Each layer has its own process group ID.

Definitions

A *name* is a sequence of characters delimited by a blank, tab or new-line. Only the first eight characters are significant. The *names* (1) through (7) cannot be used when creating a layer. They are used by *shl* when no name is supplied. They can be abbreviated to just the digit.

Commands

The following commands can be issued from the *shl* prompt level; any unique prefix is accepted:

create [*name*]

Create a layer called *name* and make it the current layer. If no argument is given, a layer is created with a name of the form *(#)* where # is the last digit of the virtual device bound to the layer. The shell prompt variable **PS1** is set to the name of the layer followed by a space. A maximum of seven layers can be created.

block *name* [*name* ...]

For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **-loblk** within the layer.

delete *name* [*name* ...]
For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal [see *signal*(2)].

help (or **?**)
Print the syntax of the *shl* commands.

layers [**-l**] [*name* ...]
For each *name*, list the layer name and its process group. The **-l** option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

resume [*name*]
Make *name* the current layer. If no argument is given, the last existing current layer is resumed.

toggle Resume the layer that was current before the last current layer.

unblock *name* [*name* ...]
For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option –**loblk** within the layer.

quit Exit *shl*. All layers are sent the SIGHUP signal.

name Make *name* the current layer.

FILES

/dev/sxt/???	Virtual tty devices
$SHELL	Variable containing the path name of the shell to use (default is /bin/sh).
/etc/drvload	

SEE ALSO

sh(1), stty(1). ioctl(2), signal(2), sxt(7).

NAME

showmount - show all remote mounts

SYNOPSIS

/etc/showmount [**-a**] [**-d**] [**-e**] [**-r**] [host]

DESCRIPTION

The *showmount* command lists all clients that have remotely mounted a filesystem from *host*. This information is maintained by *mountd*(1M) on the server *host*, and is saved by the server across crashes in the file **/etc/rmtab**. The default value for *host* is the Internet hostname name returned by *hostname*(1).

OPTIONS

-d List directories that have been remotely mounted by clients.

-a Print all remote mounts in the format

hostname:directory

where *hostname* is the name of the client, and *directory* is the root of the file system that has been mounted.

-e Print the list of exported file systems and eligible clients. For the purpose of mounting, the eligible client list includes all hostname aliases.

-r Remove all references to this client in the **/etc/rmtab** of the server *host*. Used by the client upon rebooting after a crash; restricted to the super-user.

SEE ALSO

rmtab(4), mountd(1M), exports(4).

BUGS

If a client crashes, its entry is not removed from the server's list until it reboots and goes multi-user.

NAME

shutdown, halt - shut down system, change system state

SYNOPSIS

/etc/shutdown [**-y**] [**-g***grace_period*] [**-i***init_state*]

/etc/halt

DESCRIPTION

The *shutdown* command is executed by the super-user to change the state of the machine. By default, it brings the system to "single-user" state.

The command sends a warning message and a final message before it starts actual shutdown activities. By default, the command asks for confirmation before it starts shutting down daemons and killing processes. The options are used as follows:

-y Pre-answers the confirmation question so the command can be run without user intervention. A default of 60 seconds is allowed between the warning message and the final message. Another 60 seconds is allowed between the final message and the confirmation.

-g*grace_period*
Allows the super-user to change the number of seconds from the 60-second default.

-i*init_state*
Specifies the state that *init*(1M) is to be put in following the warnings, if any. By default, system state "**s**" is used. (Currently, all single user states execute **/etc/rc0**, which kills all processes, unmounts all file systems except root, and spawns a shell at the terminal that executed *shutdown* or *halt*).

After entering single user state, the system can be prepared for power off by using **/etc/reboot -h**.

The *halt* command shuts down CTIX in a safe but abrupt way. It is meant for small installations where verbal warnings are faster than terminal messages.

DIAGNOSTICS

The most common error diagnostic to occur is *device busy*. This diagnostic happens when a particular file system could not be unmounted.

SEE ALSO

init(1M), rc0(1M), rc2(1M), inittab(4).
S/Series CTIX Administrator's Guide.

NAME

size - print section sizes in bytes of common object files

SYNOPSIS

size [**-n**] [**-f**] [**-o**] [**-x**] [**-V**] files

DESCRIPTION

The *size* command produces section size information in bytes for each loaded section in the common object files. The size of the text, data, and bss (uninitialized data) sections is printed, as well as the sum of the sizes of these sections. If an archive file is input to the *size* command the information for all archive members is displayed.

The **-n** option includes NOLOAD sections in the size.

The **-f** option produces full output, that is, it prints the size of every loaded section, followed by the section name in parentheses.

Numbers will be printed in decimal unless either the **-o** or the **-x** option is used, in which case they will be printed in octal or in hexadecimal, respectively.

The -V flag will supply the version information on the *size* command.

SEE ALSO

as(1), cc(1), ld(1), a.out(4), ar(4).

CAVEAT

Since the size of bss sections is not known until link-edit time, the *size* command will not give the true total size of pre-linked objects.

DIAGNOSTICS

size: name: cannot open
if *name* cannot be read.

size: name: bad magic
if *name* is not an appropriate common object file.

NAME

slattach, sldetach - attach and detach serial lines as network interfaces

SYNOPSIS

/etc/slattach devname source destination [baudrate]

/etc/sldetach interface-name

DESCRIPTION

slattach is used to assign a serial (tty) line to a network interface using the DARPA Internet Protocol, and to define the source and destination network addresses. The *devname* parameter is the name of the device the serial line is attached to, for example, **/dev/tty001**. The source and destination are either host names present in the host name data base [see *hosts*(4)], or DARPA Internet addresses expressed in the Internet standard "dot notation". The optional *baudrate* parameter is used to set the speed of the connection; if not specified, the default of 9600 is used.

Only the superuser may attach or detach a network interface.

sldetach is used to remove the serial line that is being used for IP from the network tables and allow it to be used as a normal terminal again. *Interface-name* is the name that is shown by *netstat*(1).

EXAMPLES

```
/etc/slattach tty001 tom-src genstar
/etc/slattach /dev/tty001 hugo dahl 4800
/etc/sldetach sl0
```

FILES

/etc/hosts
/dev/*

DIAGNOSTICS

Various messages indicating:
- the specified interface does not exist
- the requested address is unknown
- the user is not the superuser

SEE ALSO

hosts(4), netstat(1), ifconfig(1M).

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

alarm(2), sleep(3C).

NAME

slipd - switched Serial Line Internet Protocol control facility

SYNOPSIS

/etc/slipd [**-kufsvd**] [**-e** timeout] [**-l** logfile]
/etc/slipin

DESCRIPTION

The *slipd* facility controls the CTIX Switched SLIP (Serial Line Internet Protocol) facility. This facility permits the use of switched asynchronous communication links, such as telephones and dataswitches, for CTIX Internetworking. All actions performed by *slipd* are recorded in a log file. The default log file is **/etc/log/sliplog**.

slipd runs in three modes: master mode, slave mode, and control mode. Each mode is discussed separately below.

- **Master Mode.** In master mode, *slipd* has three basic functions:

  1. Maintain the kernel table of switched SLIP links.
  2. Take connection requests (via the kernel) and make connections. After making the connection, *slipd* performs the same functions as *slattach*(1M).
  3. Clean up when connections are done. This involves monitoring the phone line, and performing the functions of *sldetach* [see *slattach*(1M)].

  Normally switched SLIP uses the same **Devices** file as UUCP, **/usr/lib/uucp/Devices**. However, switched SLIP uses a different **Systems** file, **/usr/lib/uucp/Systems.slip**, to keep connection information. The information is in the same format as the equivalent UUCP files. Note that **Systems.slip** is associated with the *slip* service by the file **/usr/lib/uucp/Sysfiles**.

  Connections to be made via switched SLIP are requested in the same manner as other CTIX network connections [using *listen*(2) and *connect*(2)]. User requests are recognized using special routing table entries, and are passed to the master daemon.

  slipd in master mode is usually invoked at boot time via **/etc/init.d/devices** (if there is a file named **/etc/rcopts/SLIPD**).

slipd is affected by the following flags:

-d operate in debug mode. Status of connections being made is written to the log.

-e [*timeout*]
change timeouts, where *timeout* is specified in seconds. This option is used while logging into the remote system and making connections. Default is 15 seconds.

-v write log output to *stderr*, as well as to the log file.

-l *logfile*
use file name supplied as *logfile* rather than the default **/etc/log/sliplog**.

- **Slave Mode.** *slipd* in slave mode is usually invoked as a login shell in **/etc/passwd** (via the link **/etc/slipin**). An example of such an entry follows:

```
slip:qNJEagn2oFrIc:50:50:Switched SLIP slave:/usr/spool/uucp:/etc/slipin
```

Once invoked, *slipd* does the following:

1. Finds out from the master who the caller is and validates the login.
2. Performs an *slattach*(1M).
3. Monitors the line and performs an *sldetach* [see *slattach*(1M)].

Authentication is perfomed in a manner similar to UUCP, using UUCP's Permissions file, **/usr/lib/uucp/Permissions**. Currently only the LOGNAME entry is used.

Slave mode is entered automatically when *slipd* is invoked as *slipin*. However, it may also be invoked using the **-s** flag.

- **Control Mode.** Control mode is used to send messages to the master *slipd*. Control mode is invoked by specifying one of the following flags:

-u immediately update kernel routing tables from the system host database. (This is done automatically every 180 minutes.)

-f flush all present connections.

-k kill the master daemon.

-d toggle daemon's debug mode.

Monitoring SLIP

Two tools are available for monitoring Switched SLIP:

1. The log file. The log file records the following information:
   - when requests for connections are received
   - any errors during establishment of the connection
   - when connections are closed
   - any Control Mode actions
   - connection status when the **-d** flag is specified to the daemon
   - the associations between *sw* and *sl* devices (see below).
2. The **netstat**(1) command. **netstat -r** shows which hosts will presently be connected to using switched SLIP. Routes with interfaces of type *sw* are switched.

Switched links will show up in **netstat -i** as inactive (for example, *sw0*)* while not connected. After connection the *sw* device will be active, and the new *sl* (slip) device will appear with address *Any* (because the interface address is associated with the *sw* device).

FILES

/etc/passwd	contains SLIP logins
/etc/inittab	specifies uugetty(1M) for communication lines
/etc/gettydefs	line speed definitions used in inittab
/etc/log/sliplog	default log file
/usr/spool/locks/slippid.*	PIDs of daemons
/usr/lib/uucp/Sysfiles	points to Systems file
/usr/lib/uucp/Systems.slip	default Systems file
/usr/lib/uucp/Permissions	Permissions file

SEE ALSO

netstat(1), slattach(1M), uucico(1M).

NAME

slink, ldsocket - STREAMS linker, load socket configuration

SYNOPSIS

slink [**-v**] [**-c** cfile]

ldsocket [**-dvw**] [**-c** cfile]

DESCRIPTION

slink and *ldsocket* are used to initialize the CTIX STREAMS-based internetworking software. They are normally run at boot time and are called from the script **/etc/init.d/devices** (which has links to files in **/etc/rc?.d** directories). Note that **/etc/init.d/devices** invokes *slink* if it finds a file named **/etc/rcopts/KSTRM**, and invokes *ldsocket* if it finds a file named **/etc/rcopts/KSOCK**.

slink is the STREAMS linker. It links the available STREAMS protocols modules into the running system. *slink* must be run before most networking facilities become available. *slink* reads the **/etc/netcf** file to obtain configuration information, and remains as a daemon process to maintain the linkage.

ldsocket initializes the CTIX Berkeley networking compatability interface, which is an alternate stream head supporting the *socket*(2) system call family. *ldsocket* loads the kernel with associations between the protocol family, type and number triplets passed to the *socket* system call, and the STREAMS devices supporting those protocols. *ldsocket* reads the **/etc/netcf** file to obtain configuration information, and must be run before the Berkeley networking interface can be used.

The following options are recognized by both *slink* and *ldsocket*:

-c *cfile* Use *cfile* instead of **/etc/netcf**

-v Verbose mode

The following options are recognized by *slink* only:

-w Print warning messages

-d Print debugging messages

FILES

/etc/netcf

SEE ALSO

netcf(4), intro(7).

NAME

sno - SNOBOL interpreter

SYNOPSIS

sno [files]

DESCRIPTION

sno is a SNOBOL III compiler and interpreter (with slight differences). *sno* obtains input from the concatenation of the named *file*s and the standard input. All input through a statement containing the label **end** is considered program and is compiled. The rest is available to **syspit**.

sno differs from SNOBOL III in the following ways:

There are no unanchored searches. To get the same effect:

```
a ** b          unanchored search for b.
a *x* b = x c   unanchored assignment
```

There is no back referencing.

```
x = "abc"
a *x* x         is an unanchored search for abc.
```

Function declaration is done at compile time by the use of the (non-unique) label **define**. Execution of a function call begins at the statement following the **define**. Functions cannot be defined at run time, and the use of the name **define** is preempted. There is no provision for automatic variables other than parameters. Examples:

```
define f( )
define f(a, b, c)
```

All labels except **define** (even **end**) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on **end** cannot merely name a label.

If **start** is a label in the program, program execution will start there. If not, execution begins with the first executable statement; **define** is not an executable statement.

There are no built-in functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators / and * must be set off by spaces. The right side of assignments must be non-empty. Either ´ or " may be used for literal quotes.

The pseudo-variable **sysppt** is not available.

SEE ALSO

awk(1).

NAME

sort - sort and/or merge files

SYNOPSIS

sort [**-cmu**] [-ooutput] [**-y**kmem] [**-z**recsz] [**-dfiMnr**] [**-btx**] [+pos1 [-pos2]] [files]

DESCRIPTION

sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if - is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The following options alter the default behavior:

-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.

-u Unique: suppress all but one in each set of lines having equal keys.

-o*output*

The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **-o** and *output*.

-y*kmem*

The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, **-y**0 is guaranteed to start with minimum memory. By convention, **-y** (with no argument) starts with maximum memory.

-z*recsz*

The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **-c** or **-m** options, a popular system default size will be used. Lines

longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules.

-d "Dictionary" order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

-f Fold lower case letters into upper case.

-i Ignore non-printable characters.

-M Compare as months. The first three non-blank characters of the field are folded to upper case and compared. For example, in English the sorting order is "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The **-M** option implies the **-b** option (see below).

-n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **-n** option implies the **-b** option (see below). Note that the **-b** option is only effective when restricted sort key specifications are in effect.

-r Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation *+pos1 -pos2* restricts a sort key to one beginning at *pos1* and ending just before *pos2*. The characters at position *pos1* and just before *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *-pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

-b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **-b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *-pos2* argument (see below).

-tx Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (for example, *xx* delimits an empty field).

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfinr**. A starting position specified by *+m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m*+1st field; *+m*.0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by *-m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*+1st field; *-m*.1**b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 -2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort -r -o outfile +1.0 -1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

```
sort -r +1.0b -1.1b infile1 infile2
```

Print the password file [*passwd*(4)] sorted by the numeric user ID (the third colon-separated field):

```
sort -t: +2n -3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **-um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort -um +2 -3 infile
```

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

WARNINGS

Comments and exits with non-zero status for various trouble conditions (for example, when input lines are too long), and for disorder discovered under the **-c** option. When the last line of an input file is missing a **new-line** character, *sort* appends one, prints a warning message, and continues.

sort does not guarantee preservation of relative line ordering on equal keys.

NAME

spell, hashmake, spellin, hashcheck - find spelling errors

SYNOPSIS

spell [**-v**] [**-b**] [**-x**] [**-l**] [**-i**] [+local_file] [files]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

The *spell* program collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable from (by applying certain inflections, prefixes, and/or suffixes) words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

The *spell* program ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

Invoked with the **-v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Invoked with the **-b** option, British spelling is checked. Besides preferring such spellings as *centre*, *colour*, *programme*, *speciality*, *travelled*, this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

Invoked with the **-x** option, every plausible stem is printed with = for each word.

By default, *spell* [like *deroff*(1)] follows chains of included files [**.so** and **.nx** *troff*(1) requests], *unless* the names of such included files begin with **/usr/lib**. Invoked with the **-l** option, *spell* follows the chains of *all* included files. Invoked with the **-i** option, *spell* ignores all chains of included files.

Invoked with the *+local_file* option, words found in *local_file* are removed from *spell*'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources and, while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files can be specified by specifying name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (for example, thier=thy-y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell*:

hashmake Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.

spellin Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.

hashcheck Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/usr/lib/spell/spellhist	history file
/usr/lib/spell/spellprog	program

SEE ALSO

deroff(1), eqn(1), sed(1), sort(1), tbl(1), tee(1), troff(1).

BUGS

The spelling list's coverage is uneven; new installations will probably want to monitor the output for several months to gather local additions; typically, additions are kept in a separate local file that is added to the hashed *spelling_list* by *spellin*.

NAME

spline - interpolate smooth curve

SYNOPSIS

spline [options]

DESCRIPTION

spline takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *graph*(1G).

The following *options* are recognized, each as a separate argument:

-a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

-k The constant k used in the boundary value computation:

$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$

is set by the next argument (default $k = 0$).

-n Space output points so that approximately n intervals occur between the lower and upper x limits (default $n = 100$).

-p Make output periodic, that is, match derivatives at ends. First and last input values should normally agree.

-x Next 1 (or 2) arguments are lower (and upper) x limits. Normally, these limits are calculated from the data. Automatic abscissas start at lower limit (default 0).

SEE ALSO

graph(1G).

DIAGNOSTICS

When data is not strictly monotone in x, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1,000 input points is enforced silently.

NAME

split - split a file into pieces

SYNOPSIS

split [*-n*] [file [name]]

DESCRIPTION

split reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically, up to **zz** (a maximum of 676 files). *Name* cannot be longer than 12 characters. If no output name is given, **x** is default.

If no input file is given, or if - is given in its stead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

NAME

starter - information about the operating system for beginning users

SYNOPSIS

[help] starter

DESCRIPTION

The CTIX system Help Facility command *starter* provides five categories of information about the CTIX system to assist new users.

The five categories are:

- commands a new user should learn first
- CTIX system documents important for beginners
- education centers offering CTIX system courses
- local environment information
- on-line teaching aids installed on the CTIX system

The user may choose one of the above categories by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit"). When a category is chosen, the user will receive one or more pages of information pertaining to it.

From any screen in the Help Facility, a user may execute a command via the shell [*sh*(1)] by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your *.profile* file [see *profile* (4)]:

```
export SCROLL ; SCROLL=no
```

If you later decide that scrolling is desired, **SCROLL** must be set to **yes**.

Information on each of the Help Facility commands (*starter*, *locate*, *usage*, *glossary*, and *help*) is located on their respective manual pages.

SEE ALSO

glossary(1), help(1), locate(1), sh(1), usage(1), term(5).

WARNINGS

If the shell variable **TERM** [see *sh*(1)] is not set in the user's *.profile* file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5).

NAME

stat - statistical network useful with graphical commands

SYNOPSIS

node-name [options] [files]

DESCRIPTION

The *stat* utility is a collection of command level functions (nodes) that can be interconnected using *sh*(1) to form a statistical network. The nodes reside in **/usr/bin/graf** (see *graphics*(1G)). Data is passed through the network as sequences of numbers (vectors), where a number of the following form is evaluated in the usual way:

[sign](digits)(.digits)[**e**[sign]digits]

Brackets and parentheses surround fields. All fields are optional, but at least one of the fields surrounded by parentheses must be present. Any character input to a node that is not part of a number is taken as a delimiter.

Note that *stat* nodes are divided into four classes:

Transformers	Map input vector elements into output vector elements.
Summarizers	Calculate statistics of a vector.
Translators	Convert among formats.
Generators	Sources of definable vectors.

Most nodes accept options indicated by a leading minus (-). In general, an option is specified by a character followed by a value, such as **c5**. This is interpreted as **c** := 5 (**c** is assigned 5). The following keys are used to designate the expected type of the value:

c	characters
i	integer
f	floating point or integer
file	file name
string	string of characters, surrounded by quotes to include a *shell* argument delimiter

Options without keys are flags. All nodes except *generators* accept files as input.

Below is a list of synopses for *stat* nodes:

Transformers:

abs [**-c***i*] - absolute value
columns (similarly for **-c** options that follow)

af [**-c***i* **t v**] - arithmetic function
titled output, **v**erbose

ceil [**-c***i*] - round up to next integer

cusum [**-c***i*] - cumulative sum

exp [**-c***i*] - exponential

floor [**-c***i*] - round down to next integer

gamma [**-c***i*] - gamma

list [**-c***i* **d***string*] - list vector elements
delimiter(s)

log [**-c***i* **b***f*] - logarithm
base

mod [**-c***i* **m***f*] - modulus
modulus

pair [**-c***i* **F***file* **x***i*] - pair elements
File containing base vector, **x** group size

power [**-c***i* **p***f*] - raise to a power
power

root [**-c***i* **r***f*] - take a root
root

round [**-c***i* **p***i* **s***i*] - round to nearest integer, .5 rounds to 1
places after decimal point, **s**ignificant digits

siline [**-c***i* **i***f* **n***i***s***f*] - generate a line given slope and intercept
intercept, **n**umber of positive integers, **s**lope

sin [**-c***i*] - sine

subset [**-a***f* **b***f* **c***i* **F***file* **i***i* **l***f* **n***l* **n p** *p f* **s***i* **t***i*] - generate a subset
above, **b**elow, **F**ile with master vector, **i**nterval, **l**eave, master contains element **n**umbers to **l**eave, master contains element **n**umbers to **p**ick, **p**ick, **s**tart, **t**erminate

Summarizers:

bucket [**-a***i* **c***i* **F***file* **h***f* **i***i* **l***f* **n***i*] - break into buckets
average size, **F**ile containing bucket boundaries, **h**igh, **i**nterval, **l**ow, **n**umber
Input data should be sorted

cor [**-F***file*] - correlation coefficient
File containing base vector

hilo [**- h l o ox oy**]- find high and low values
high only, **l**ow only, **o**ption form, **o**ption form with **x** prepended, **o**ption form with **y** prepended

lreg [**-F***file* **i o s**] - linear regression
File containing base vector, **i**ntercept only, **o**ption form for *siline*, **s**lope only

mean [**-f***f* **n***i* **p***f*] - (trimmed) arithmetic mean
fraction, **n**umber, **p**ercent

point [**-f***f* **n***i* **p***f* **s**] - point from empirical cumulative density function
fraction, **n**umber, **p**ercent, **s**orted input

prod - internal product

qsort [**-c***i*] - quick sort

rank - vector rank

total - sum total

var - variance

Translators:

bar [**-a b f g r***i* **w***i* **x***f* **xa y***f* **ya yl***f* **yh***f*] - build a bar chart
suppress **a**xes, **b**old, suppress **f**rame, suppress **g**rid, **r**egion, **w**idth in percent, **x** origin, suppress **x**-axis label, **y** origin, suppress **y**-axis label, **y**-axis **l**ower bound, **y**-axis **h**igh bound
Data is rounded off to integers.

hist [**-a b f g r***i* **x***f* **xa y***f* **ya yl***f* **yh***f*] - build a histogram
suppress **a**xes, **b**old, suppress **f**rame, suppress **g**rid, **r**egion, **x** origin, suppress **x**-axis label, **y** origin, suppress **y**-axis label, **y**-axis **l**ower bound, **y**-axis **h**igh bound

label [**-b c F***file* **h p r***i* **x xu y yr**] - label the axis of a GPS file
bar chart input, retain **c**ase, **l**abel **F**ile, **h**istogram input, **p**lot input, **r**otation, **x**-axis, **u**pper **x**-axis, **y**-axis, **r**ight **y**-axis

pie [**-b o p pn***i* **pp***i* **r***i* **v x***i* **y***i*] - build a pie chart
bold, values **o**utside pie, value as **p**ercentage(:=100), value as **p**ercentage(:=**i**), draw **p**ercent of **p**ie, **r**egion, no values, **x** origin, **y** origin
Unlike other nodes, input is lines of the form
[**< i e f c***c* **>**] value [label]
ignore (do not draw) slice, **e**xplode slice, **f**ill slice, **c**olor slice *c*=(**b**lack, **r**ed, **g**reen, **b**lue)

plot [**-a b c***string* **d f F***file* **g m r***i* **x***f* **xa x***i**f* **xh***f* **xl***f* **xn***i* **xt y***f* **ya y***i**f* **yh***f* **yl***f* **yn***i* **yt**] - plot a graph
suppress **a**xes, **b**old, plotting **c**haracters, **d**isconnected, suppress **f**rame, **F**ile containing x vector, suppress **g**rid, **m**ark points, **r**egion, **x** origin, suppress **x**-axis label, **x** interval, **x** **h**igh bound, **x** low bound, **n**umber of ticks on **x**-axis, suppress **x**-axis **t**itle, **y** origin, suppress **y**-axis label, **y** interval, **y** **h**igh bound, **y** low bound, **n**umber of ticks on **y**-axis, suppress **y**-axis **t**itle

title [**-b c l***string* **v***string* **u***string*] - title a vector or a GPS
title **b**old, retain **c**ase, **l**ower title, **u**pper title, vector title

Generators:

gas [**-c***i* **i***f* **n***i* **s***f* **t***f*] - generate additive sequence
interval, **n**umber, **s**tart, **t**erminate

prime [**-c***i* **h***i* **l***i* **n***i*] - generate prime numbers
high, **l**ow, **n**umber

rand [**-c***i* **h***f* **l***f* **m***f* **n***i* **s***i*] - generate random sequence
high, **l**ow, **m**ultiplier, **n**umber, **s**eed

RESTRICTIONS

Some nodes have a limit on the size of the input vector.

SEE ALSO

graphics(1G), gps(4).

NAME

strclean - STREAMS error logger cleanup program

SYNOPSIS

strclean [**-d** logdir] [**-a** age]

DESCRIPTION

The *strclean* program is used to clean up the STREAMS error logger directory on a regular basis [for example, by using *cron*(1M)]. By default, all files with names matching **error.*** in **/usr/adm/streams** that have not been modified in the last three days are removed. The **-d** options specifies a directory other than **/usr/adm/streams**. The **-a** options specifies the maximum age in days for a log file.

EXAMPLE

The following command produces the same result as running *strclean* with no arguments:

strclean -d /usr/adm/streams -a 3

NOTES

The *strclean* program is typically run from *cron*(1M) on a daily or weekly basis.

FILES

/usr/adm/streams/error.*

SEE ALSO

cron(1M), strerr(1M).
UNIX System V Release 3.2 Streams Programmer's Guide.

NAME

strerr - STREAMS error logger daemon

SYNOPSIS

strerr

DESCRIPTION

The *strerr* daemon receives error log messages from the STREAMS log driver [*log*(7)] and appends them to a log file. The error log files produced reside in the directory **/usr/adm/streams**, and are named **error.***mm-dd*, where *mm* is the month and *dd* is the day of the messages contained in each log file.

The format of an error log message follows:

<seq> <time> <ticks> <flags> <mid> <sid> <text>

<seq>	error sequence number
<time>	time of message in hh:mm:ss
<ticks>	time of message in machine ticks since boot priority level
<flags>	T : the message was also sent to a tracing process F : indicates a fatal error N : send mail to the system administrator
<mid>	module ID number of source
<sid>	sub-ID number of source
<text>	formatted text of the error message

Messages that appear in the error log are intended to report exceptional conditions that require the attention of the system administrator. Those messages that indicate the total failure of a STREAMS driver or module should have the F flag set. Those messages requiring the immediate attention of the administrator will have the N flag set, which causes the error logger to send the message to the system administrator through *mail*(1). The priority level usually has no meaning in the error log but has meaning if the message is also sent to a tracer process.

Once initiated, *strerr* continues to execute until terminated by the user. Commonly, *strerr* is executed asynchronously.

CAVEATS

Only one *strerr* process at a time is permitted to open the STREAMS log driver.

If a module or driver is generating a large number of error messages, running the error logger causes a degradation in STREAMS performance. If a large burst of messages are generated in a short time, the log driver may not be able to

deliver some of the messages; this situation is indicated by gaps in the sequence numbering of the messages in the log files.

FILES

/usr/adm/streams/error.*mm-dd*

SEE ALSO

log(7).
UNIX System V Release 3.2 Streams Programmer's Guide.

NAME

strings - extract the ASCII text strings in a file

SYNOPSIS

strings [**-a**] [**-o**] [**-#**] file ...

DESCRIPTION

strings looks for ASCII text strings in a file. It is useful for examining and identifying object and other binary files. A string is any sequence of 4 or more printing characters ending with a newline or a null. If the file is an object file, the search is restricted to the initialized data space.

Here are the options:

-a Don't restrict object file searches.

-o Precede each string with its octal offset.

-# Make # the minimum string length instead of 4.

SEE ALSO

od(1).

WARNING

The algorithm for identifying strings is rather primitive.

NAME

strip - strip symbol and line number information from a common object file

SYNOPSIS

strip [**-l**] [**-x**] [**-i**] [**-r**] [**-V**] filename ...

DESCRIPTION

The *strip* command strips the symbol table and line number information from common object files, including archives. Once this has been done, no symbolic debugging access will be available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using any of the following options:

-l Strip line number information only; do not strip any symbol table information.

-x Do not strip static or external symbol information.

-b Same as the **-x** option, but also do not strip scoping information (for example, beginning and end of block delimiters).

-r Do not strip static or external symbol information, or relocation information.

-V Print the version of the strip command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, *strip* will complain and terminate without stripping *filename* unless the **-r** option is used.

If the *strip* command is executed on a common archive file [see *ar*(4)] the archive symbol table will be removed. The archive symbol table must be restored by executing the *ar*(1) command with the s option before the archive can be link-edited by the *ld*(1) command. *strip* will produce appropriate warning messages when this situation arises.

strip is used to reduce the file storage overhead taken by the object file.

FILES

TMPDIR/strp* temporary files

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

SEE ALSO

ar(1), as(1), cc(1), ld(1), tmpnam(3S), a.out(4), ar(4).

DIAGNOSTICS

strip: name: cannot open
if *name* cannot be read.

strip: name: bad magic
if *name* is not an appropriate common object file.

strip: name: relocation entries present; cannot strip
if *name* contains relocation entries and the **-r** flag is not used, the symbol table information cannot be stripped.

NAME

stty - set the options for a terminal

SYNOPSIS

stty [**-a**] [**-g**] [options]

DESCRIPTION

The *stty* command sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

In this report, if a character is preceded by a caret (^), the value of that option is the corresponding Control character (for example, ^H is Control-H; in this case, recall that Control-H is the same as the Backspace key). The sequence ^^ means that an option has a null value. For example, normally **stty -a** reports that the value of **swtch** is ^^; however, if **shl**(1) has been invoked, **stty -a** has the value ^z.

-a Reports all option settings.

-g Reports current settings in a form that can be used as an argument to another *stty* command.

Options in the last group are implemented by using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (**-parenb**)
Enable (disable) parity generation and detection.

parodd (**-parodd**)
Select odd (even) parity.

cs5 cs6 cs7 cs8
Select character size [see *termio*(7)].

0 Hang up phone line immediately.

110 300 600 1200 1800 2400 4800 9600 19200 38400
Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)

hupcl (**-hupcl**)
Hang up (do not hang up) Dataphone connection on last close.

hup (**-hup**)
Same as **hupcl** (**-hupcl**).

cstopb (-cstopb)
Use two (one) stop bits per character.

cread (-cread)
Enable (disable) the receiver.

clocal (-clocal)
Assume a line without (with) modem control.

ctscd (-ctscd)
Enable (disable) CTS as transmit enable

Input Modes

ignbrk (-ignbrk)
Ignore (do not ignore) break on input.

brkint (-brkint)
Signal (do not signal) INTR on break.

ignpar (-ignpar)
Ignore (do not ignore) parity errors.

parmrk (-parmrk)
Mark (do not mark) parity errors [see *termio*(7)].

inpck (-inpck)
Enable (disable) input parity checking.

istrip (-istrip)
Strip (do not strip) input characters to seven bits.

inlcr (-inlcr)
Map (do not map) NL to CR on input.

igncr (-igncr)
Ignore (do not ignore) CR on input.

icrnl (-icrnl)
Map (do not map) CR to NL on input.

iuclc (-iuclc)
Map (do not map) uppercase alphabetics to lowercase on input.

ixon (-ixon)
Enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (-ixany)
Allow any character (only DC1) to restart output.

ixoff (-ixoff)
Request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (-opost)
Post-process output (do not post-process output; ignore all other output modes).

olcuc (-olcuc)
Map (do not map) lowercase alphabetics to uppercase on output.

onlcr (-onlcr)
Map (do not map) NL to CR-NL on output.

ocrnl (-ocrnl)
Map (do not map) CR to NL on output.

onocr (-onocr)
Do not (do) output CRs at column zero.

onlret (-onlret)
On the terminal NL performs (does not perform) the CR function.

ofill (-ofill)
Use fill characters (use timing) for delays.

ofdel (-ofdel)
Fill characters are DELs (NULs).

cr0 cr1 cr2 cr3
Select style of delay for carriage returns [see *termio*(7)].

nl0 nl1 Select style of delay for line-feeds [see *termio*(7)].

tab0 tab1 tab2 tab3
Select style of delay for horizontal tabs [see *termio*(7)].

bs0 bs1 Select style of delay for backspaces [see *termio*(7)].

ff0 ff1 Select style of delay for form-feeds [see *termio*(7)].

vt0 vt1 Select style of delay for vertical tabs [see *termio*(7)].

Local Modes

isig (-isig)
Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.

icanon (-icanon)
Enable (disable) canonical input (ERASE and KILL processing).

xcase (-xcase)
Canonical (unprocessed) uppercase/lowercase presentation.

echo (-echo)
Echo back (do not echo back) every character typed.

echoe (-echoe)
Echo (do not echo) ERASE character as a backspace-space-backspace string. Note that this mode erases the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, can be confusing on escaped characters, tabs, and backspaces.

echok (-echok)
Echo (do not echo) NL after KILL character.

lfkc (-lfkc)
The same as **echok (-echok)**; obsolete.

echonl (-echonl)
Echo (do not echo) NL.

noflsh (-noflsh)
Disable (enable) flush after INTR, QUIT, or SWTCH.

Control Assignments

control-character c
Set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **swtch**, **eof**, **ctab**, **min**, or **time** [**min** and **time** are used with **-icanon**; see *termio*(7)]. If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding Control character (for example, ^D is Control-D); ^? is interpreted as DEL; and ^- is interpreted as undefined.

line *i* Set line discipline to *i* ($0 < i < 127$).

Combination Modes

evenp or **parity**
Enable **parenb** and **cs7**.

oddp Enable **parenb**, **cs7**, and **parodd**.

-parity, **-evenp**, or **-oddp**
Disable **parenb**, and set **cs8**.

raw (-raw or **cooked)**
Enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).

nl (**-nl**) Unset (set) **icrnl**, **onlcr**. In addition **-nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

lcase (**-lcase**)
Set (unset) **xcase**, **iuclc**, and **olcuc**.

LCASE (-LCASE)
Same as **lcase** (**-lcase**).

tabs (**-tabs** or **tab3**)
Preserve (expand to spaces) tabs when printing.

ek Reset ERASE and KILL characters back to normal # and @.

sane Resets all modes to some reasonable values.

term Set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

Cluster Terminals

Options that are meaningless to the RS-422 interface are ignored by cluster terminals. See *termio*(7) for specifics.

SEE ALSO

tabs(1), ioctl(2), termio(7).

NAME

su - become super-user or another user

SYNOPSIS

su [-] [name [arg ...]]

DESCRIPTION

su allows one to become another user without logging off. The default user *name* is **root** (that is, super-user).

To use *su*, the appropriate password must be supplied (unless one is already **root**). If the password is correct, *su* will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry [see *passwd*(4)], or **/bin/sh** if none is specified [see *sh*(1)]. To restore normal user ID privileges, type an **EOF** (*cntrl-d*) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like *sh*(1), an *arg* of the form **-c** *string* executes *string* via the shell and an arg of **-r** will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(1). If the first argument to *su* is a -, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is -, thus causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of **$PATH**, which is set to **/bin:/etc:/usr/bin** for **root**. Note that if the optional program used as the shell is **/bin/sh**, the user's **.profile** can check *arg0* for **-sh** or **-su** to determine if it was invoked by *login*(1) or *su*(1), respectively. If the user's program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *-program* by both *login*(1) and *su*(1).

All attempts to become another user using *su* are logged in the log file **/usr/adm/sulog**.

EXAMPLES

To become user **bin** while retaining your previously exported environment, execute:

```
su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, execute:

```
su - bin
```

To execute *command* with the temporary environment and permissions of user **bin**, type:

```
su - bin -c "command args"
```

FILES

/etc/passwd	system's password file
/etc/profile	system's profile
$HOME/.profile	user's profile
/usr/adm/sulog	log file

SEE ALSO

env(1), login(1), sh(1), passwd(4), profile(4), environ(5).

NAME

sum - print checksum and block count of a file

SYNOPSIS

sum [**-r**] file

DESCRIPTION

sum calculates and prints a 16-bit checksum for the named file, and also prints the number of 512-byte blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **-r** causes an alternate algorithm to be used in computing the checksum.

SEE ALSO

wc(1).

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NAME

swap - swap administrative interface

SYNOPSIS

/etc/swap -a swapdev [swaplow [swaplen]]

/etc/swap -d swapdev [swaplow]

/etc/swap -l

DESCRIPTION

The *swap* command provides a method of adding, deleting, and monitoring the system swap areas used by the memory manager. The following options are recognized:

-a Add the specified swap area. *Swapdev* is the name of block special device: for example, **/dev/dsk/c0d1s2**. *Swaplow* is the offset, in 512-byte blocks, into the device where the swap area should begin. *Swaplen* is the length of the swap area, in 512-byte blocks, up to the size of the specified partition. Note that this option can be used only by the super-user. Swap areas are added at system startup time through one or more entries in **/etc/rcopts/KSWAP**: each entry consists of the *swapdev* to be added.

-d Delete the specified swap area. *Swapdev* is the name of block special device: for example, **/dev/dsk/c0d1s2**. *Swaplow* is the offset, in 512-byte blocks, into the device where the swap area should begin. Using this option marks the swap area as INDEL (in the process of being deleted). The system does not allocate any new blocks from the area, and tries to free swap blocks from it. The area remains in use until all blocks from it are freed. Note that this option can be used only by the super-user.

-l List the status of all swap areas. The output has five columns:

path	The *swapdev* special file for the swap area, if one can be found in the **/dev/dsk** or **/dev** directories.
dev	The major/minor device number of the *swapdev*, in decimal.
swaplow	The *swaplow* value for the area, in 512-byte blocks.
blocks	The *swaplen* value for the area, in 512-byte blocks.

free The number of free 512-byte blocks in the area. If the swap area is being deleted, this column is marked INDEL.

SEE ALSO

S/Series CTIX Administrator's Guide.

WARNINGS

No check is performed to see if a swap area being added overlaps with an existing swap area or file system.

NAME

sync - update the super block

SYNOPSIS

sync

DESCRIPTION

sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync*(2) for details.

NOTE

If you have done a write to a file on a remote machine in a Remote File Sharing environment, you cannot use *sync* to force buffers to be written out to disk on the remote machine. *sync* will only write local buffers to local disks.

SEE ALSO

sync(2).

NAME

sysdef - output system definition

SYNOPSIS

/etc/sysdef [namelist [master]]

DESCRIPTION

sysdef outputs the current system definition in tabular form. It lists all hardware devices, their local bus addresses, and unit count, as well as pseudo devices, system devices, loadable modules and the values of all tunable parameters. It generates the output by analyzing the named operating system file (*namelist*) and extracting the configuration information from the name list itself.

FILES

/unix	default operating system file (where the system namelist is)
/etc/master	default master device information table

SEE ALSO

master(4), nlist(3C).

DIAGNOSTICS

internal name list overflow
: if the master table contains more than an internally specified number of entries for use by *nlist*(3C).

NAME

tabs - set tabs on a terminal

SYNOPSIS

tabs [tabspec] [**-T**type] [**+mn**]

DESCRIPTION

The *tabs* command sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

tabspec Four types of tab specification are accepted for *tabspec*. They are described below: canned (*-code*), repetitive (*-n*), arbitrary (*n1,n2,...*), and file (*--file*). If no *tabspec* is given, the default value is **-8**, that is, CTIX system "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, for example, the DASI 300, DASI 300s, and DASI 450.

-code Use one of the codes listed below to select a *canned* set of tabs. The legal codes and their meanings are as follows:

-a 1,10,16,36,72
Assembler, IBM S/370, first format

-a2 1,10,16,40,72
Assembler, IBM S/370, second format

-c 1,8,12,16,20,55
COBOL, normal format

-c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows [see *fspec*(4)]:

```
<:t-c2 m6 s66 d:>
```

-c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54, 58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than **-c2.** This is the recommended format for COBOL. The appropriate format specification is [see *fspec*(4)]:

```
<:t-c3 m6 s66 d:>
```

-f 1,7,11,15,19,23
FORTRAN

-p 1,5,9,13,17,21,25,29,33,37,41,45,49,53, 57,61
PL/I

-s 1,10,55
SNOBOL

-u 1,12,20,44
UNIVAC 1100 Assembler

-n A *repetitive* specification requests tabs at columns 1+*n*, 1+2**n*, etc. Of particular importance is the value **8**: this represents the CTIX system ''standard'' tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value **0**, implying no tabs at all.

n1, *n2*, ... The *arbitrary* format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats **1,10,20,30**, and **1,10,+10,+10** are considered identical.

--file If the name of a *file* is given, *tabs* reads the first line of the file, searching for a format specification [see *fspec*(4)]. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(1) command:

tabs -- file; pr file

Any of the following also may be used; if a given flag occurs more than once, the last value given takes effect:

-T*type* *tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in *term*(5). If no **-T** flag is supplied, *tabs* uses the value of the environment variable **TERM**. If **TERM** is not defined in the *environment* [see *environ*(5)], *tabs* tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is **10**.

For a TermiNet, the first value in the tab list should be **1**, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

EXAMPLES

The following command uses *-code* (*canned* specification) to set tabs to the settings required by the IBM assembler: columns 1, 10, 16, 36, 72.

tabs -a

The following command uses *-n* (*repetitive* specification), where *n* is **8**, setting tabs every eighth position: 1+(1*8), 1+(2*8), ..., which evaluate to columns 9, 17, and so on.

tabs -8

The following command uses *n1,n2, ...* (*arbitrary* specification) to set tabs at columns 1, 8, and 36.

tabs 1,8,36

The following command uses *--file* (*file* specification) to indicate that tabs should be set according to the first line of *$HOME/fspec.list/att4425* [see *fspec*(4)].

tabs --$HOME/fspec.list/att4425

DIAGNOSTICS

illegal tabs
Arbitrary tabs are ordered incorrectly.

illegal increment
A zero or missing increment is found in an arbitrary specification.

unknown tab code
A *canned* code cannot be found.

can't open
The file named in the *--file* option cannot be opened.

file indirection
The specification in the file named in the *--file* option points to yet another file. Indirection of this form is not permitted.

SEE ALSO

newform(1), pr(1), tput(1), fspec(4), terminfo(4), environ(5), term(5).

NOTE

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

The *tabs* command clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

WARNING

The *tabspec* used with the *tabs* command is different from that used with the *newform*(1) command. For example, **tabs -8** sets tabs every eighth position; however, **newform -i-8** indicates that tabs are set every eighth position.

NAME

talk - talk to another user

SYNOPSIS

talk person [ttyname]

DESCRIPTION

The *talk* program copies lines from your terminal to that of another user.

If you want to talk to someone on your own machine, *person* is just the person's login name. If you want to talk to a user on another host, *person* is of one of the following forms:

host!user
host.user
host:user
user@host (This is the preferred form.)

If you want to talk to a user who is logged in more than once, the *ttyname* argument can be used to indicate the appropriate terminal name.

When *talk* is first invoked, it sends the following message to the user you want to talk to:

Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine

At this point, the recipient of the message should reply as follows:

talk your_name@your_machine

It does not matter which machine the recipient uses for the reply, as long as the user's login-name is the same. Once communication is established, the two parties can type simultaneously, with their output appearing in separate windows. The CTRL-L key sequence reprints the screen; erase, kill, and word kill characters work in *talk* as normal. Use the interrupt character to exit *talk*; *talk* then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk can be denied or granted by use of the *mesg*(1) 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), named(1M), talkd(1M), who(1), mail(1), write(1).

BUGS

This version of *talk* uses a protocol that is compatible with 4.3BSD. It is incompatible with other vendors' 4.2BSD versions of *talk*.

NAME

talkd - remote user communication server

SYNOPSIS

/etc/talkd

DESCRIPTION

talkd is the server that notifies a user that somebody else wants to initiate a conversation. It acts a repository of invitations, responding to requests by clients wishing to rendezvous to hold a conversation. In normal operation, a client, the caller, initiates a rendezvous by sending a CTL_MSG to the server of type LOOK_UP (see *<protocols/talkd.h>*). This causes the server to search its invitation tables to check if an invitation currently exists for the caller (to speak to the callee specified in the message). If the lookup fails, the caller then sends an ANNOUNCE message causing the server to broadcast an announcement on the callee's login ports requesting contact. When the callee responds, the local server uses the recorded invitation to respond with the appropriate rendezvous address and the caller and callee client programs establish a stream connection through which the conversation takes place.

talkd is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

SEE ALSO

talk(1), write(1), inetd.conf(4).

NAME

tapeset - set drive parameters for tape controllers

SYNOPSIS

tapeset [**-p**] device

tapeset [**-t** drive_type] [**-f** tapedrives_file] [**-p**] device

DESCRIPTION

The *tapeset* command initializes VME controller-based half-inch and SCSI controller-based tape drives. Normally initialization is performed automatically through entries in **/etc/drvload** and does not need to be repeated until the system is rebooted. (The **/etc/drvload** file looks for the presence of **/etc/rcopts/KIPT** and **/etc/rcopts/KSCSI**; if they exist, they are "dotted;" see the FILES section.)

Execute *tapeset* after loading the appropriate tape driver [by using *lddrv*(1M)]. *Device* is **/dev/rmt/c?d#c**, where # is the drive number. (The second **c** indicates to the driver software that this special controller command does not require that the drive be online.)

The *tapeset* command recognizes the following options:

-f *tapedrives_file* Use the file *tapedrives_file* instead of **/etc/tapedrives** for tape drive-specific information.

-p Print the current configuration on **stdout**.

-t *drive_type* Automatically configure the controller for a *drive_type* drive. Examine the **/etc/tapedrives** file for a list of supported tape drives.

EXAMPLES

The following command displays the current configuration of VME controller-based half-inch tape drive 0:

tapeset -p /dev/rmt/c1d0c

The following command enables autoload on SCSI controller-based tape drive 0 (S/640 only):

tapeset -t 5945S-auto /dev/rmt/c0d0c

The following command sets the drive parameters of drive 1 for a Cipher M990 tape drive and then displays the resulting configuration:

tapeset -t M990 -p /dev/rmt/c1d1c

FILES

/dev/rmt/c?d#c

/etc/tapedrives

/etc/rcopts/KIPT contains tapeset commands for half-inch drives and causes automatic execution of these commands when the system is rebooted

/etc/rcopts/KSCSI contains tapeset commands for SCSI tape drives and causes automatic execution of these commands when the system is rebooted; note that on S/640 systems, if this file does not exist, the following tapeset command is issued by default:

tapeset -t 5945S-noauto /dev/rmt/c0d0c

SEE ALSO

config(1M), lddrv(1M), tapedrives(4), ipt(7).
MightyFrame VME Half-Inch Tape Controller Card Manual.

WARNINGS

Some SCSI half-inch tape drives do not accept commands if the drive is not online; you may have to perform *tapeset* commands by hand after the system is booted.

NAME

tar - tape file archiver

SYNOPSIS

/etc/tar -c[**vwfb**[*#s*]] *device block files ...*

/etc/tar -r[**vwb**[*#s*]] *device block* [*files ...*]

/etc/tar -t[**vf**[*#s*] *device*

/etc/tar -u[**vwb**[*#s*]] *device block* [*files ...*]

/etc/tar -x[**lmovwf**[*#s*]] *device* [*files ...*]

DESCRIPTION

The *tar* command saves and restores files on magnetic tape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing one function letter (c, r, t, u, or x) and possibly followed by one or more function modifiers (v, w, f, b, and #). Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r Replace. The named *files* are written on the end of the tape. The **c** function implies this function. Blocked tapes cannot be appended.

x Extract. The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. Use the file or directory's relative path when appropriate, or *tar* will not find a match. The owner, modification time, and mode are restored (if possible). If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.

t Table. The names and other information for the specified files are listed each time that they occur on the tape. The listing is similar to the format produced by the *ls*(1) command; if the **v** option is used with **t**, the listing produced is like that of the *ls*-l command. If no *files* argument is given, all the names on the tape are listed.

u Update. The named *files* are added to the tape if they are not already there, or have been modified since last written on that tape. This key implies the **r** key. Blocked tapes (including QIC tapes) cannot be updated.

c Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This key implies the **r** key.

The following characters can be used in addition to the letter that selects the desired function; use them in the order shown in the synopsis.

#s This modifier determines the drive on which the tape is mounted (replace # with the drive number) and the speed of the drive (replace s with **l**, **m**, or **h** for low, medium or high). The modifier tells *tar* to use a drive other than the default drive, or the drive specified with the **-f** option. For example, with the **5h** modifier, *tar* would use **/dev/rmt/c0d5h** instead of the default drive **/dev/rmt/c0d0**. The default is **0**. The density option is ignored on some tapes, such as QIC tapes.

v Verbose. Normally, *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w What. This causes *tar* to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with **y** is given, the action is performed. Any other input means "no". This is not valid with the **t** key.

f File. This causes *tar* to use the *device* argument as the name of the archive instead of **/dev/rmt/c0d0?** or **/dev/mt/c0d0?** (where ? is [**lmh**]). If the name of the file is **-**, *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *tar* can also be used to move hierarchies with the following command:

cd fromdir; tar cf - . | (cd todir; tar xf -)

b Blocking Factor. This causes *tar* to use the *block* argument as the blocking factor for tape records (512 bytes). The default is 128 for most tape drives; the maximum is 128. A warning message appears if your device does not handle 64K bytes. This function should not be supplied when operating on regular archives or block special devices. It is mandatory however, when reading archives on raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes created on block special devices (key letters **x** and **t**).

l Link. This tells *tar* to complain if it cannot resolve all of the links to the files being dumped. If **l** is not specified, no error messages are printed.

m Modify. This tells *tar* to not restore the modification times. The modification time of the file will be the time of extraction.

o Ownership. This causes extracted files to take on the user and group identifier of the user running the program, rather than those on tape. This is only valid with the **x** key.

The following command can be used to archive onto a QIC tape:

cd dir; tar c

FILES

/dev/rmt/*
/dev/mt/*
/tmp/tar*

SEE ALSO

ar(1), cpio(1), ls(1), ipt(7), qic(7).

DIAGNOSTICS

Complains about bad key characters and tape read/write errors.

Complains if enough memory is not available to hold the link tables.

NOTES

CTIX does not currently support block tape devices. Specifying the block device (for example, **/dev/mt/c0d0**) will cause *tar* to fail.

BUGS

There is no way to ask for the *n*-th occurrence of a file.

Tape errors are handled ungracefully.

The **u** option can be slow.

The **b** option should not be used with archives that are going to be updated. The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the **b** option should not be used at all, because updating an archive stored on disk can destroy it.

The current limit on file name length is 100 characters.

tar doesn't copy empty directories or special files.

The **r** option does not work with QIC tapes.

NAME

tdl, gtdl, ptdl - RS-232 terminal download

SYNOPSIS

/usr/local/bin/tdl [type]
/usr/local/bin/gtdl [runfile]
/usr/local/bin/ptdl [runfile]

DESCRIPTION

tdl, gtdl, and *ptdl* download a terminal system image over a an RS-232 line. The program is run from the terminal that is to receive the system image, which must be a Convergent Technologies terminal running in boot ROM emulation mode.

Type is a number that specifies one of the standard terminal types. If *type* is omitted, *tdl* sends an escape sequence to the terminal to discover its type. (If the user has not used the boot ROM **T** command, the escape sequence produces a "101" on a Programmable Terminal and a "201" on a Graphics Terminal. These cause *tdl* to download **/usr/lib/iv/ws101.232** or **/usr/lib/iv/ws201.232**, respectively.)

Runfile is the name of a download image file.

ptdl and *gtdl* require a terminal with a Release 1.0 boot ROM. *tdl* requires a terminal with a Release 2.0 boot ROM.

To use *tdl*, follow this procedure:

1. Turn the terminal on while holding down the space bar. Be sure to keep the space bar down until the boot ROM prompt appears on the screen.
2. Use the boot ROM commands to set whatever communication options you need. Do not use the **T** (system image type) command unless you need a nonstandard type.
3. Enter the boot ROM **E** (emulate serial terminal) command.
4. If necessary, establish a connection with the host system and log in as *tdl, ptdl,* or *gtdl.*
5. Run *tdl, ptdl,* or *gtdl* with no parameters.

To allow users to download their terminals by logging in, for example, as **tdl,** add the appropriate login entries to **/etc/passwd**:

```
tdl::50:1:Terminal Down Load:/:/usr/local/bin/tdl
ptdl::51:1:PT232 Download (1.0 boot ROM) :/:/usr/local/bin/ptdl
gtdl::52:1:GT232 Download (1.0 boot ROM) :/:/usr/local/bin/gtdl
```

The download area must be specified on the disk; see *iv*(1).

FILES

/usr/lib/iv/ws*.232	CTIX copies of the system images
/usr/local/bin/ws*.232	checked if system image not in /usr/lib/iv

When acting on a type sent from the terminal, *tdl* downloads **/usr/liv/iv/wsxxx.232**, where *xxx* is the three-digit terminal type. If that file is missing, *tdl* looks for **/usr/local/bin/wsxxx.232.**

SEE ALSO

iv(1).
Programmable Terminal Programmer's Guide.
Graphics Terminal Programmer's Guide.

DIAGNOSTICS

The terminal displays dashs (-) to indicate successfully transmitted blocks, questions marks (?) to indicate nonfatal tranmission errors. A fatal transmission error produces an appropriate message from the terminal and a return to the boot ROM emulate code; you may need to press the RETURN key to get a shell prompt.

WARNINGS

tdl, *tdtl*, and *ptdl* do not verify that the download file is a valid terminal system image.

The 2.0 GT boot ROM does not support downloading run images greater than 65,536 bytes. Attempting to download images greater than 65,536 bytes may cause the terminal to fail.

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*. There are two options:

-i ignores interrupts;

-a causes the output to be appended to the *files* rather than overwriting them.

NAME

telnet - user interface to TELNET protocol

SYNOPSIS

/usr/local/bin/telnet [host]

DESCRIPTION

The *telnet* command establishes connections to other hosts using the TELNET protocol. It is more general than *rlogin*(1) because TELNET servers run under a wider variety of operating systems. However, *rlogin* is more convenient to use.

Establishing a Single Connection

If *host* is specified, *telnet* establishes a connection to that host; *host* can be a host name or a DARPA Internet address in dot notation [see *hosts*(4)]. While the connection remains open, *telnet* is in input mode (see below). When the connection is closed, *telnet* terminates. Usually the remote system closes the connection when you log out. To close the connection yourself, use the escape character to enter the **close** command (see below).

Input Mode

The *telnet* command enters input mode when a connection is opened and exits it when a connection is closed. In input mode all text typed goes to the remote host except when the escape character is typed.

To enter a single *telnet* command without first closing the connection, press the escape character at any time in input mode. Initially the escape character is control-[(ASCII GS; octal 035); The *telnet* prompt (**telnet>**) appears, and *telnet* executes a single command line instead of sending it to the remote host. After you press Return and the command is executed, *telnet* resumes sending your input to the remote host, unless your command closed the connection (**close** or **quit**).

Use the **escape** command to change the escape character.

The input mode entered is either "character-at-a-time" or "line-by-line," depending on what the remote system supports.

In character-at-a-time mode, most text typed is immediately sent to the remote host for processing.

In line-by-line mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The local echo character (initially **E**) can be used to disable and enable the local echo.

In either mode, if the *localchars* toggle is TRUE (the default in line mode; see below), the user's *quit*, *intr*, and *flush* characters are trapped locally, and sent as TELNET protocol sequences to the remote side. Some options (see **toggle**

autoflush and **toggle** *autosynch* below) cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of *quit* and *intr*).

Command Mode

If *host* is not specified, *telnet* enters command mode. The *telnet* prompt appears, and *telnet* understands the following commands (and truncated command names, as long as they aren't ambiguous; for example, **ope** is valid, but **op** is not):

? [*command*] Give command summaries. If *command* is specified, give just that command summary.

AO Send the TELNET command **AO** (abort output) and an out-of-band signal to the remote server with **DM** as the synchronizing mark.

AYT Send the TELNET command **AYT** (are you there?) and an out-of-band signal to the remote server with **DM** as the synchronizing mark.

BREAK Send the TELNET command **BREAK** to the remote server.

EC Send the TELNET command **EC** (erase character) to the remote server.

EL Send the TELNET command **EL** (erase line) to the remote server.

IP Send the TELNET command **IP** (interrupt process) and an out-of-band signal to the remote server with **DM** as the synchronizing mark.

SYNCH Send an out-of-band signal to the remote server with **DM** as the synchronizing mark.

crmod Turn on/off carriage return mode. Initially carriage return mode is off. When carriage return mode is on, carriage return characters from the remote host are expanded to a carriage return followed by a line feed.

close Close the current connection. Useful only with the escape character (see "Input Mode" above).

display [*argument...*]
Display all, or some, of the **set** and **toggle** values (see below).

do *option* Tell the remote server to process *option*. This command is used largely for testing option negotiation.

dont *option* Tell the remote server to stop processing *option*. This command is used largely for testing option negotiation.

escape Change the escape character used in input mode (see below). The *telnet* command prompts for a new escape character; press a key that generates a single character, then press Return. To leave the escape character unchanged, press Return without entering a character.

help [*command*]

Give summaries of commands. If *command* is specified, give summary of just that command.

mode *type*

Type is either *line* (for line-by-line mode) or *character* (for character-at-a-time mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode is entered.

open *host* Open a connection to *host*. While the connection remains open, *telnet* is in input mode (see below). If you close the connection with a *telnet* command from input mode, *telnet* returns to command mode; if the connection is closed from the other end, *telnet* terminates. Usually the remote system closes the connection when you log out.

options Turn on/off viewing of TELNET options negotiations. Initially viewing is off. When viewing is on, *telnet* shows its negotiations with the *telnetd*.

quit Close any open connection and terminate *telnet*.

send *arguments* Sends one or more special character sequences to the remote host. The following arguments can be specified; more than one argument can be specified at one time:

escape Sends the current *telnet* escape character (initially ^]).

synch Sends the **TELNET SYNCH** sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system -- if it doesn't work, a lower case **r** can be echoed on the terminal).

brk Sends the **TELNET BRK** (break) sequence, which can have significance to the remote system.

ip Sends the **TELNET IP** (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

ao Sends the **TELNET AO** (Abort Output) sequence, which should cause the remote system to flush all output **from** the remote system **to** the user's terminal.

ayt Sends the **TELNET AYT** (Are You There) sequence, to which the remote system may or may not choose to respond.

ec Sends the **TELNET EC** (Erase Character) sequence, which should cause the remote system to erase the last character entered.

el Sends the **TELNET EL** (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

ga Sends the **TELNET GA** (Go Ahead) sequence, which likely has no significance to the remote system.

nop Sends the **TELNET NOP** (No OPeration) sequence.

set *argument value*

Set any one of a number of *telnet* variables to a specific value. The special value **off** disables the function associated with the variable. The values of variables may be interrogated with the **display** command. Valid values follow:

echo This is the value (initially ^E) which, when in line-by-line mode, toggles between local echoing of entered characters (for normal processing) and suppressed echoing of entered characters (for entering, say, a password).

escape This is the *telnet* escape character (initially ^[), which causes entry into *telnet* command mode (when connected to a remote system).

interrupt

If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the interrupt character is typed, a

TELNET **IP** sequence (see **send** *ip* above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's **intr** character.

quit If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *quit* character is typed, a TELNET **BRK** sequence (see **send** *brk* above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's **quit** character.

flushoutput
If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *flushoutput* character is typed, a TELNET **AO** sequence (see **send** *ao* above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's **flush** character.

erase If *telnet* is in *localchars* mode (see **toggle** *localchars* below), **and** if *telnet* is operating in character-at-a-time mode, when this character is typed, a TELNET **EC** sequence (see **send** *ec* above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's **erase** character.

kill If *telnet* is in *localchars* mode (see **toggle** *localchars* below), **and** if *telnet* is operating in character-at-a-time mode, when this character is typed, a TELNET **EL** sequence (see **send** *el* above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's **kill** character.

eof If *telnet* is operating in line-by-line mode, entering this character as the first character on a line sends this character to the remote system. The initial value of the EOF character is taken to be the terminal's EOF character.

status Show the current connection and escape character.

toggle *arguments* ...
Toggle (between TRUE and FALSE) various flags that control how *telnet* responds to events. More than one argument may

be specified. The state of these flags may be interrogated with the **display** command. Valid arguments follow:

localchars

If this is TRUE, the *flush*, *interrupt*, *quit*, *erase*, and *kill* characters (see **set** above) are recognized locally, and transformed into (hopefully) appropriate **TELNET** control sequences (respectively *ao*, *ip*, *brk*, *ec*, and *el*; see **send** above). The initial value for this toggle is TRUE in line-by-line mode, and FALSE in character-at-a-time mode.

autoflush

If *autoflush* and *localchars* are both TRUE, then when the *ao*, *intr*, or *quit* characters are recognized (and transformed into **TELNET** sequences; see **set** above for details), *telnet* refuses to display any data on the user's terminal until the remote system acknowledges (via a **TELNET** *Timing Mark* option) that it has processed those **TELNET** sequences. The initial value for this toggle is TRUE if the terminal user had not performed an **stty noflsh**; otherwise FALSE [see *stty(1)*].

autosynch

If *autosynch* and *localchars* are both TRUE; and then when either the *intr* or *quit* characters is typed (see **set** above) the resulting **TELNET** sequence sent is followed by the **TELNET SYNCH** sequence. This procedure **should** cause the remote system to begin throwing away all previously typed input until both of the **TELNET** sequences have been read and acted upon. The initial value of this toggle is FALSE.

crmod Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host is mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is FALSE.

debug Toggles socket level debugging (useful only to the *super*user). The initial value for this toggle is FALSE.

options Toggles the display of some internal *telnet* protocol processing (having to do with **TELNET** options). The initial value for this toggle is FALSE.

netdata Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

? Displays the legal **toggle** commands.

will *option* Tell the remote server we will process *option*. This command is used largely for testing option negotiation.

wont *option* Tell the remote server we won't process *option*. This command is used largely for testing option negotiation.

z Suspend *telnet*. This command works only when the user is using the *csh*(1).

escape character Send the escape character to the remote host.

Telnet Options

Once a connection is established, both sides negotiate various options to get the best possible service. The following options are recognized:

BINARY Controls transmission of binary data.

ECHO Controls echoing.

SGA Suppress go ahead.

STATUS Status of options.

TM Timing Mark.

EXOPL Extended Options List.

SEE ALSO

rlogin(1), telnetd(1M).

BUGS

There is no adequate way for dealing with flow control.

On some remote systems, echo must be manually disabled in line-by-line mode.

There is enough settable state to justify a *.telnetrc* file.

No capability for a *.telnetrc* file is provided.

In line-by-line mode, the terminal's EOF character is recognized (and sent to the remote system) only when it is the first character on a line.

NAME

telnetd - DARPA TELNET protocol server

SYNOPSIS

/etc/telnetd [**-d**] [port]

DESCRIPTION

The *telnetd* server supports the DARPA standard TELNET virtual terminal protocol. The TELNET server operates at the port indicated in the "telnet" service description; see *services*(4). This port number can be overridden (for debugging purposes) by specifying a port number on the command line. If the **-d** option is specified, each socket created by *telnetd* has debugging enabled [see SO_DEBUG in *socket*(2)].

The *telnetd* server operates by allocating a virtual-terminal device [see *vt*(7)] for a client, then creating a login process which has the slave side of the pseudoterminal as *stdin*, *stdout*, and *stderr*. *telnetd* manipulates the master side of the pseudoterminal, implementing the TELNET protocol and passing characters between the client and login process.

When a TELNET session is started up, *telnetd* sends a TELNET option to the client side indicating a willingness to do "remote echo" of characters. The pseudoterminal allocated to the client is configured to operate in "cooked" mode and with XTABS and CRMOD enabled [see *tty*(7)]. Aside from this initial setup, the only mode changes *telnetd* carries out are those required for echoing characters at the client side of the connection.

The following options are recognized:

BINARY	Controls transmission of binary data.
ECHO	Controls echoing.
SGA	Suppress go ahead.
STATUS	Status of options.
TM	Timing Mark.
EXOPL	Extended Options List.

The *telnetd* server is started by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

SEE ALSO

inetd(1M), telnet(1), inetd.conf(4).

NAME

test - condition evaluation command

SYNOPSIS

test expr

[expr]

DESCRIPTION

test evaluates the expression *expr* and, if its value is true, sets a zero (true) exit status; otherwise, a non-zero (false) exit status is set; *test* also sets a non-zero exit status if there are no arguments. When permissions are tested, the effective user ID of the process is used.

All operators, flags, and brackets (brackets used as shown in the second SYNOPSIS line) must be separate arguments to the *test* command; normally these items are separated by spaces.

The following primitives are used to construct *expr*:

-r *file*	true if *file* exists and is readable.
-w *file*	true if *file* exists and is writable.
-x *file*	true if *file* exists and is executable.
-f *file*	true if *file* exists and is a regular file.
-d *file*	true if *file* exists and is a directory.
-c *file*	true if *file* exists and is a character special file.
-b *file*	true if *file* exists and is a block special file.
-p *file*	true if *file* exists and is a named pipe (fifo).
-u *file*	true if *file* exists and its set-user-ID bit is set.
-g *file*	true if *file* exists and its set-group-ID bit is set.
-k *file*	true if *file* exists and its sticky bit is set.
-s *file*	true if *file* exists and has a size greater than zero.
-t [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
-z *s1*	true if the length of string *s1* is zero.
-n *s1*	true if the length of the string *s1* is non-zero.
s1 = *s2*	true if strings *s1* and *s2* are identical.

s1 **!=** *s2* true if strings *s1* and *s2* are *not* identical.

s1 true if *s1* is *not* the null string.

n1 **-eq** *n2* true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, and **-le** may be used in place of **-eq**.

These primaries may be combined with the following operators:

! unary negation operator.

-a binary *and* operator.

-o binary *or* operator (**-a** has higher precedence than **-o**).

(expr) parentheses for grouping. Notice also that parentheses are meaningful to the shell and, therefore, must be quoted.

SEE ALSO

find(1), sh(1).

WARNING

If you test a file you own (the *-r*, *-w*, or *-x* tests), but the permission tested does not have the *owner* bit set, a non-zero (false) exit status will be returned even though the file may have the *group* or *other* bit set for that permission. The correct exit status will be set if you are super-user.

The = and != operators have a higher precedence than the **-r** through **-n** operators, and = and != always expect arguments; therefore, = and != cannot be used with the **-r** through **-n** operators.

If more than one argument follows the **-r** through **-n** operators, only the first argument is examined; the others are ignored, unless a **-a** or a **-o** is the second argument.

NAME

tftp - user interface to the DARPA TFTP protocol

SYNOPSIS

tftp [host [port]]

DESCRIPTION

The *tftp* program is a user interface to the DARPA standard Trivial File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which *tftp* is to communicate can be specified on the command line, in which case *tftp* immediately attempts to establish a connection to a TFTP server on that host. Otherwise, *tftp* enters its command interpreter and awaits instructions from the user. When *tftp* is awaiting commands from the user, the following prompt appears:

tftp>

The following commands are recognized by *tftp*:

connect *host-name* [*port*]

Set the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections betweeen transfers; thus, the *connect* command does not actually create a connection, but merely remembers what host is to be used for transfers. You need not 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* can be one of *ascii* or *binary*. The default is *ascii*.

put *file*

put *localfile remotefile*

put *file1 file2 ... fileN remote-directory*

Put a file or set of files to the specified remote file or directory. Because *tftpd* allows only publicly readable and writable files to be accessed, the remote file must exist and be writable. The 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. If the remote-directory form is used, the remote host is assumed to be a *UNIX* or *CTIX* machine. The use of *tftp* does not require an account or

password on the remote system. Due to the lack of authentication information, the *tftpd* server allows only publicly readable files to be accessed.

get *filename*
get *remotename localname*
get *file1 file2 ...fileN*
Get 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. The use of *tftp* does not require an account or password on the remote system. Due to the lack of authentication information, the *tftpd* server allows only publicly readable files to be accessed.

quit Exit *tftp*. An end of file also exits.

verbose Toggle verbose mode.

trace Toggle packet tracing.

status Show current status.

rexmt *retransmission-timeout*
Set the per-packet retransmission timeout, in seconds.

timeout *total-transmission-timeout*
Set the total transmission timeout, in seconds.

ascii Shorthand for "mode ascii"

binary Shorthand for "mode binary"

? [*command-name* ...]
Print help information.

FILES

/etc/hosts

SEE ALSO

tftpd(1M).

WARNINGS

Because there is no user-login or validation within the *TFTP* protocol, the remote site usually has some sort of file-access restrictions in place. The exact methods are specific to each site.

NAME

tftpd - DARPA Trivial File Transfer Protocol server

SYNOPSIS

/etc/tftpd [**-d**] [port]

DESCRIPTION

tftpd is a server that supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the "tftp" service description; see *services*(4). This port number may be overridden (for debugging purposes) by specifying a port number on the command line. If the **-d** option is specified, each socket created by *tftpd* will have debugging enabled [see SO_DEBUG in *socket*(2)].

The use of *tftp* does not require an account or password on the remote system. Due to the lack of authentication information, *tftpd* allows only publicly readable and writable files to be accessed. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling tftp service.

tftpd is spawned by the "super-server" *inetd*, and therefore must have an entry in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)]. Note that the *tftpd* entry in this file must be "wait": this is to avoid subsequent *selects* from being successful before the first *tftpd* process does its *receive*. *tftpd* takes care to prevent multiple *tftpd* processes from being spawned to service the same request. (*inetd* is able to continue processing new messages on the port.)

SEE ALSO

inetd(1M), adman(1), tftp(1), inetd.conf(4), services(4).

WARNINGS

This server is known only to be self consistent [that is, it operates with the user TFTP program, *tftp*(1)].

The search permissions of the directories leading to the files accessed are not checked.

NAME

tic - terminfo compiler

SYNOPSIS

tic [**-v**[n]] [**-c**] file

DESCRIPTION

The *tic* command translates a *terminfo*(4) file from the source format into the compiled format. The results are placed in the directory */usr/lib/terminfo*. The compiled format is necessary for use with the library routines described in *curses*(3X).

-vn (Verbose) output to standard error trace information showing *tic*'s progress. The optional integer *n* is a number from 1 to 10, inclusive, indicating the desired level of detail of information. If *n* is omitted, the default level is 1. If *n* is specified and greater than 1, the level of detail is increased.

-c Check only *file* for errors. Errors in **use=** links are not detected.

file Contains one or more *terminfo*(4) terminal descriptions in source format [see *terminfo*(4)]. Each description in the file describes the capabilities of a particular terminal. When a **use=***entry-name* field is discovered in a terminal entry currently being compiled, *tic* reads in the binary from */usr/lib/terminfo* to complete the entry. (Entries created from *file* will be used first. If the environment variable **TERMINFO** is set, that directory is searched instead of */usr/lib/terminfo*.) *tic* duplicates the capabilities in *entry-name* for the current entry, with the exception of those capabilities that explicitly are defined in the current entry.

If the environment variable **TERMINFO** is set, the compiled results are placed there instead of */usr/lib/terminfo*.

FILES

/usr/lib/terminfo/?/* compiled terminal description data base

SEE ALSO

curses(3X), term(4), terminfo(4).
UNIX System V Release 3.2 Programmer's Guide.

WARNINGS

Total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

Terminal names exceeding 14 characters are truncated to 14 characters and a warning message is printed.

When the -c option is used, duplicate terminal names are not diagnosed; however, when -c is not used, they are diagnosed.

BUGS

To allow executables from previous releases of CTIX to run with the compiled terminfo entries created by the new terminfo compiler, canceled capabilities are not marked as canceled within the terminfo binary unless the entry name has a plus sign (+) within it. (Such terminal names are only used for inclusion within other entries through the use of a **use=** entry; such names would not be used for real terminal names.)

For example:

```
4415+nl, kf1@, kf2@, ....

4415+base, kf1=\EOc, kf2=\EOd, ....

4415-nl|4415 terminal without keys,
        use=4415+nl, use=4415+base,
```

The above example works as expected; the definitions for the keys do not show up in the *4415-nl* entry. However, if the entry *4415+nl* did not have a plus sign within its name, the cancelations would not be marked within the compiled file and the definitions for the function keys would not be canceled within *4415-nl*.

DIAGNOSTICS

Most diagnostic messages produced by *tic* during the compilation of the source file are preceded with the approximate line number and the name of the terminal currently being worked on.

mkdir ... returned bad status
: The named directory could not be created.

File does not start with terminal names in column one
: The first thing seen in the file, after comments, must be the list of terminal names.

Token after a seek(2) not NAMES
: Somehow the file being compiled changed during the compilation.

Not enough memory for use_list element
or
Out of memory
: Not enough free memory was available (*malloc*(3) failed).

Can't open ...
: The named file could not be created.

Error in writing ...
The named file could not be written to.

Can't link ... to ...
A link failed.

Error in re-reading compiled file ...
The compiled file could not be read back in.

Premature EOF
The current entry ended prematurely.

Backspaced off beginning of line
This error indicates something wrong happened within *tic*.

Unknown Capability - "..."
The named invalid capability was found within the file.

Wrong type used for capability "..."
For example, a string capability was given a numeric value.

Unknown token type
Tokens must be followed by '@' to cancel, ',' for booleans, '#' for numbers, or '=' for strings.

"...": bad term name
or
Line ...: Illegal terminal name - "..."
Terminal names must start with a letter or digit
The given name was invalid. Names must not contain white space or slashes, and must begin with a letter or digit.

"...": terminal name too long.
An extremely long terminal name was found.

"...": terminal name too short.
A one-letter name was found.

"..." filename too long, truncating to "..."
The given name was truncated to 14 characters due to CTIX file name length limitations.

"..." defined in more than one entry. Entry being used is "...". An entry was found more than once.

Terminal name "..." synonym for itself
A name was listed twice in the list of synonyms.

At least one synonym should begin with a letter.
At least one of the names of the terminal should begin with a letter.

Illegal character - "..."
The given invalid character was found in the input file.

Newline in middle of terminal name
The trailing comma was probably left off of the list of names.

Missing comma
A comma was missing.

Missing numeric value
The number was missing after a numeric capability.

NULL string value
The proper way to say that a string capability does not exist is to cancel it.

Very long string found. Missing comma?
Self-explanatory.

Unknown option. Usage is:
An invalid option was entered.

Too many file names. Usage is:
Self-explanatory.

"..." non-existant or permission denied
The given directory could not be written into.

"..." is not a directory
Self-explanatory.

"...": Permission denied
Access denied.

"...": Not a directory
tic wanted to use the given name as a directory, but it already exists as a file.

SYSTEM ERROR!! Fork failed!!!
A *fork*(2) failed.

Error in following up use-links. Either there is a loop
in the links or they reference non-existent terminals.
The following is a list of the entries involved:
A *terminfo*(4) entry with a **use**=*name* capability either referenced a non-existant terminal called *name* or *name* somehow referred back to the given entry.

NAME

time - time a command

SYNOPSIS

time command

DESCRIPTION

The *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.

The times are printed on standard error.

SEE ALSO

times(2).

NAME

timex - time a command; report process data and system activity

SYNOPSIS

timex [options] command

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of *timex* is written on standard error.

Options are:

-p List process accounting records for *command* and all its children. Suboptions **f, h, k, m, r,** and **t** modify the data items reported. The options are as follows:

- **-f** Print the *fork/exec* flag and system exit status columns in the output.
- **-h** Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:

  (total CPU time)/(elapsed time).
- **-k** Instead of memory size, show total kcore-minutes.
- **-m** Show mean core size (the default).
- **-r** Show CPU factor (user time/(system-time + user-time).
- **-t** Show separate system and user CPU times. The number of blocks read or written and the number of characters transferred are always reported.

-o Report the total number of blocks read or written and total characters transferred by *command* and all its children.

-s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

SEE ALSO

acctcom(1), sar(1).

WARNING

Process records associated with *command* are selected from the accounting file **/usr/adm/pacct** by inference, since process genealogy is not available. Background processes having the same user-ID, terminal-ID, and execution time window will be spuriously included.

EXAMPLES

A simple example:

```
timex -ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a subshell:

```
timex -opskmt sh
        session commands
EOT
```

NAME

tio - tape io filter

SYNOPSIS

tio -r tape_device [**-b** blocksize]
tio -w tape_device [**-b** blocksize]

DESCRIPTION

tio reads from or writes to a tape device asynchronously, which results in high throughput tape streaming. If the **-r** option is used, *tio* reads from tape_device and writes to standard output; if the **-w** option is used, *tio* reads from standard input and writes to tape_device. The block size specified with **tio -r** must be the same as the block size specified with **tio -w** when the tape is made.

When end-of-tape is reached, *tio* prompts the user to choose between continuing (by inserting a new tape) or exiting. The user may select either the same tape device or a new tape device by pressing Return when the drive is ready.

The **-b** flag can be used to select a particular block size which can be specified in the same format as in *dd*(1); for example, 512 (512 bytes), 64k (64*1024 bytes), 128b (128*512 bytes). The default is 65536 bytes.

Although *tio* has been optimized to support tape streaming, the user may get only partial streaming, depending on the archiving software and tape drives used. For example, *cpio*(1) is usually too slow for *tio*, especially when there are a lot of small files. On the contrary, the Cipher 990 Caching tape drive is too fast for *tio* to stream.

For the quarter-inch cartridge tape drive, the user can expect increase in performance of about 100% using *cpio*(1) and *tio*. For the half-inch drives (Cipher 880), the user can expect a 50% performance gain if *cpio*(1) is used and about a 200% performance gain if *dd*(1) is used.

EXAMPLES

```
find . -print | cpio -oQc | tio -w /dev/rmt0

tio -r /dev/rmt0 | cpio -itcQ

dd if=/dev/rdsk/c0d1s1 bs=10k | tio -w /dev/rmt/c1d0h
```

FILES

/dev/rmt*, /dev/rmt/*

NAME

toc: dtoc, ttoc, vtoc - graphical table of contents routines

SYNOPSIS

dtoc [directory]

ttoc mm-file

vtoc [**-cdhnimsvn**] [TTOC file]

DESCRIPTION

The commands listed below reside in **/usr/bin/graf** [see *graphics*(1G)].

dtoc Dtoc makes a textual table of contents, TTOC, of all subdirectories beginning at *directory* (*directory* defaults to .). The list has one entry per directory. The entry fields from left to right are level number, directory name, and the number of ordinary readable files in the directory. *Dtoc* is useful in making a visual display of all or parts of a file system. The following command makes a visual display of all readable directories under /:

dtoc / | vtoc | td

ttoc Output is the table of contents generated by the .TC macro of *mm*(1) translated to TTOC format. The input is assumed to be an *mm* file that uses the .H family of macros for section headers. If no *file* is given, the standard input is assumed.

vtoc *Vtoc* produces a GPS describing a hierarchy chart from a TTOC. The output drawing consists of boxes containing text connected in a tree structure. If no *file* is given, the standard input is assumed. Each TTOC entry describes one box and has the followingform:

id [*line-weight,line-style*] "*text*" [*mark*]

where:

id Is an alternating sequence of numbers and dots. The *id* specifies the position of the entry in the hierarchy. The *id* **0.** is the root of the tree.

line-weight Is one of the following:

n normal-weight

m medium-weight

b bold-weight

line-style Is one of the following:

so	solid-line
do	dotted-line
dd	dot-dash line
da	dashed-line
ld	long-dashed line

text Is a character string surrounded by quotation marks. The characters between the quotation marks become the contents of the box. To include a quotation mark within a box, escape it (\").

mark Is a character string (surrounded by quotation marks if it contains spaces), with included dots being escaped. The string is put above the top right corner of the box. To include either a quotation mark or a dot within a *mark*, escape it.

Entry example: **1.1 b,da "ABC" DEF**

Entries can span more than one line by escaping the newline (**\newline**).

Comments are surrounded by the /*,*/ pair; comments can appear anywhere in a TTOC.

Options to *vtoc* follow:

c	Use text as entered (default is all upper case).
d	Connect the boxes with diagonal lines.
h*n*	Horizontal interbox space is *n*% of box width.
i	Suppress the box *id*.
m	Suppress the box *mark*.
s	Do not compact boxes horizontally.
v*n*	Vertical interbox space is *n*% of box height.

SEE ALSO

graphics(1G), gps(4), mm(5).
Programmer's Guide: CTIX Supplement.

NAME

touch - update access and modification times of a file

SYNOPSIS

touch [**-amc**] [mmddhhmm[yy]] files

DESCRIPTION

The *touch* command causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified [see *date*(1)] the current time is used. The **-a** and **-m** options cause touch to update only the access or modification times respectively (default is **-am**). The **-c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1), utime(2).

NAME

tplot - graphics filters

SYNOPSIS

tplot [**-T**terminal [**-e** raster]]

DESCRIPTION

These commands read plotting instructions [see *plot*(4)] from the standard input and in general produce, on the standard output, plotting instructions suitable for a particular *terminal*. If no *terminal* is specified, the environment parameter **$TERM** [see *environ*(5)] is used. Known *terminal*s follow:

300	DASI 300.
300S	DASI 300s.
450	DASI 450.
4014	Tektronix 4014.
gt	Convergent Graphics Terminal.
ver	Versatec D1200A. 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 **-e** option causes a previously scan-converted file *raster* to be sent to the plotter.

FILES

/usr/lib/t300
/usr/lib/t300s
/usr/lib/t450
/usr/lib/t4014
/usr/lib/tgt
/usr/lib/vplot
/usr/tmp/raster$$

SEE ALSO

plot(3X), plot(4), term(5).

NAME

tput - initialize a terminal or query terminfo database

SYNOPSIS

tput [**-T type**] capname [parms ...]

tput [**-T type**] **init**

tput [**-T type**] **reset**

tput [**-T type**] **longname**

tput -S < < file

DESCRIPTION

The *tput* command uses the *terminfo*(4) database to make the values of terminal-dependent capabilities and information available to the shell [see *sh*(1)], to initialize or reset the terminal, or return the long name of the requested terminal type. If the attribute (*cap*ability *name*) is of type string, *tput* outputs a string; if the attribute is of type integer. *tput* outputs an integer; if the attribute is of type boolean, *tput* sets the exit code (**0** for TRUE if the terminal has the capability, **1** for FALSE if it does not), and produces no output. Before using a value returned on standard output, the user should test the exit code [$?, see *sh*(1)] to be sure it is **0**. (See **EXIT CODES** and **DIAGNOSTICS** below.) For a complete list of capabilities and the *capname* associated with each, see *terminfo*(4).

-T*type* indicates the *type* of terminal. Normally this option is unnecessary, because the default is taken from the environment variable **TERM**. If **-T** is specified, then the shell variables **LINES** and **COLUMNS** and the layer size [see *layers*(1)] will not be referenced.

capname indicates the attribute from the *terminfo* (4) database.

parms If the attribute is a string that takes parameters, the arguments *parms* will be instantiated into the string. An all numeric argument will be passed to the attribute as a number.

-S Allow more than one capability per invocation of *tput*. The capabilities must be passed to *tput* from a file or from the standard input instead of from the command line. Only one *capname* is allowed per line. The **-S** option changes the meaning of the **0** and **1** boolean and string exit codes (see EXIT CODES below).

init If the *terminfo*(4) database is present and an entry for the user's terminal exists (see **-T***type*, above), the following occurs: (1) if

present, the terminal's initialization strings will be output (**is1**, **is2**, **is3**, **if**, **iprog**), (2) any delays (for example, newline) specified in the entry will be set in the tty driver, (3) tabs expansion will be turned on or off according to the specification in the entry, and (4) if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will silently be skipped.

reset Instead of putting out initialization strings, the terminal's reset strings will be output if present (**rs1**, **rs2**, **rs3**, **rf**). If the reset strings are not present, but initialization strings are, the initialization strings will be output. Otherwise, **reset** acts identically to **init**.

longname If the *terminfo*(4) database is present and an entry for the user's terminal exists (see **-T***type* above), then the long name of the terminal will be put out. The long name is the last name in the first line of the terminal's description in the *terminfo*(4) database [see *term*(5)].

EXAMPLES

The following command initializes the terminal according to the type of terminal in the environmental variable **TERM**:

tput init

This command should be included in every user's .profile after the environmental variable **TERM** has been exported, as illustrated on the *profile*(4) manual page.

The following command resets an AT&T 5620 terminal, overriding the type of terminal in the environmental variable **TERM**:

tput -T5620 reset

The following command sends the sequence to move the cursor to row **0**, column **0** (the upper-left corner of the screen, usually known as the home cursor position):

tput cup 0 0

The following command echoes the clear-screen sequence for the current terminal:

tput clear

The following command prints the number of columns for the current terminal:

tput cols

The following command prints the number of columns for the 450 terminal:

tput -T450 cols

The following command sequence sets the shell variables **bold**, to begin standout mode sequence, and **offbold**, to end standout mode sequence, for the current terminal:

bold='tput smso'
offbold='tput rmso'" This might be followed by a prompt:

echo "${bold}Please type your name: ${offbold}\c"

The following command sets an exit code to indicate if the current terminal is a hardcopy terminal:

tput hc

The following command sends the sequence to move the cursor to row 23, column 4:

tput cup 23 4

The following command prints the long name from the *terminfo* (4) database for the type of terminal specified in the environmental variable **TERM**:

tput longname

The following command accepts several *tput* instructions from standard input in one invocation: it clears the screen, moves the cursor to position 10, 10, and enables bold mode. The > character prompts for *tput* commands until the list is terminated by an exclamation point (!), on a line by itself.

tput -S <<!
> clear
> cup 10 10
> bold
> !

The following command performs *tput* instructions from the file named **teeput**:

tput -S <teeput

The commands in the **teeput** file shown below, clear the screen, move the cursor to position 10, 10, and enable bold mode.

```
clear
cup 10 10
bold
```

FILES

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/include/curses.h	*curses*(3X) header file
/usr/include/term.h	*terminfo*(4) header file
/usr/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs); for more information, see the Tabs and Initialization section of *terminfo* (4)

SEE ALSO

stty (1), tabs (1), profile(4), terminfo(4).
UNIX System V Release 3.2 Programmer's Guide.

EXIT CODES

a value of **0** is set for TRUE and **1** for FALSE, unless the **-S** option is used.

If *capname* is of type string, a value of **0** is set if the *capname* is defined for this terminal *type* (the value of *capname* is returned on standard output); a value of **1** is set if *capname* is not defined for this terminal *type* (a null value is returned on standard output).

If *capname is of type boolean or* and the **-S** option is used, a value of **0** is returned to indicate that all lines were successful. No indication of which lines failed can be given, so exit code **1** never appears. Exit codes **2**, **3**, and **4** retain their usual interpretatios.

If *capname* is of type integer, a value of **0** is always set, whether or not *capname* is defined for this terminal *type*. To determine if *capname* is defined for this terminal *type*, the user must test the value of standard output. A value of **-1** means that *capname* is not defined for this terminal *type*.

Any other exit code indicates an error; see **DIAGNOSTICS**, below.

DIAGNOSTICS

tput prints the following error messages and sets the corresponding exit codes.

Exit Code	*Error Message*
0	-1 (*capname* is a numeric variable that is not specified in the *terminfo*(4) database for this terminal type, for example, **tput -T450 lines** and **tput -T2621 xmc**)

1	no error message is printed, see **EXIT CODES**, above.
2	usage error
3	unknown terminal *type* or no *terminfo*(4) database **4** unknown *terminfo*(4) capability *capname*

NAME

tr - translate characters

SYNOPSIS

tr [**-cds**] [string1 [string2]]

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **-cds** may be used:

-c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.

-d Deletes all input characters in *string1*.

-s Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a-z] Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.

[a*n] Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

EXAMPLE

The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2
```

SEE ALSO

ed(1), sh(1), ascii(5).

BUGS

Will not handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

NAME

troff - typeset text

SYNOPSIS

troff [options] [files]

DESCRIPTION

The *troff* program formats text contained in *files* (standard input by default) for a Wang Laboratories, Inc., C/A/T phototypesetter.

If no input file is given, or if the argument - is found, *troff* reads from the standard input file. The *options*, which can appear in any order, but must appear before the *files*, follow:

-o*list* Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. 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. (See *BUGS* below.)

-n*N* Number first generated page *N*.

-s*N* Stop every *N* pages. *troff* stops the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed.

-r*aN* Set register *a* (which must have a one-character name) to *N*.

-i Read standard input after *files* are exhausted.

-q Invoke the simultaneous input-output mode of the **.rd** request.

-z Print only messages generated by **.tm** (terminal message) requests.

-m*name* Prepend to the input *files* the non-compacted (ASCII text) macro file **/usr/lib/tmac /tmac.***name*.

-c*name* Prepend to the input *files* the compacted macro files **/usr/lib/macros/cmp.[nt].[dt].***name* and **/usr/lib/macros/ucmp.[nt].***name*.

-k*name* Compact the macros used in this invocation of *troff*, placing the output in files **[dt].***name* in the current directory.

-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 it is 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.

-p*N* Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.

-g Prepare output for the Murray Hill Computation Center phototypesetter and direct it to the standard output (this option is not usable on most systems). This option is not compatible with the -s option; furthermore, when this option is invoked, all **.fp** (font position) requests (if any) in the *troff* input must come before the first break, and *no* **.tl** requests may come before the first break.

-T*name* Use font-width tables for device *name* (the font tables are found in **/usr/lib/font/***name*/*). Currently, no *name*s are supported.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/font/*	font width tables for *troff*

SEE ALSO

cw(1), eqn(1), mmt(1), nroff(1), tbl(1), tc(1), mm(5), mv(5).
Programmer's Guide: CTIX Supplement.

BUGS

troff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *troff* generates may be off by one day from your idea of what the date is.

When *troff* is used with the *-olist* option inside a pipeline [for example, with one or more of *cw*(1), *eqn*(1), and *tbl*(1)], it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

NAME

true, false - provide truth values

SYNOPSIS

true

false

DESCRIPTION

true does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true
do
        command
done
```

SEE ALSO

sh(1).

DIAGNOSTICS

true has exit status zero; *false* nonzero.

NAME

tset - set terminal, terminal interface, and terminal environment

SYNOPSIS

tset [options] [**-m** [*pseudotype*][*test speed*]:*type* ...] [*type*]

DESCRIPTION

tset initializes your terminal. Its primary use is in login scripts [see *profile*(4)] to set terminal options, terminal interface options, and environment variables. Its secondary use is to restore the terminal interface and terminal after an editor or other video program has crashed.

To restore the terminal interface and terminal, just type "tset". It may be necessary to end the command with Control-J or the NEXT key instead of the RETURN key.

To set up login initialization, construct a command with the options and arguments you need and place it in your login script.

An argument indicates a terminal type to use in place of the **TERM** environment variable. If the argument begins with a question mark, *tset* prompts you for a terminal type; if you enter a blank line, you get the type specified by the argument. Terminal type arguments (in conjunction with the **-** or **-s** options) are useful at installations where none of the terminals are permanently connected to the host.

tset accepts the following options.

- Print the terminal type. This is useful for setting the **TERM** environment variable in the *.profile* file:

```
export TERM
TERM=`tset - '?adm3a'`
```

-S Print commands that will set the **TERM** and **TERMCAP** environment variables. The value for **TERMCAP** contains a description of the terminal; this makes it unnecessary for programs to read the terminal capability file each time they start up. For example:

```
eval `tset -s '?adm3a'`
```

tset uses the **SHELL** environment variable to decide the kind of commands to print.

-S Prints values for **TERM** and **TERMCAP**. Useful only in *.login*; if you use the values to set a shell variable, you get a two-element array.

-e*c* Set the erase character to *c*. Indicate a control character with a ^. If *c* is missing, *tset* uses the value of your backspace key; this is usually control-H. You also get control-H if your terminal lacks a backspace key.

-k*c* Set the kill character to *c*. Indicate a control character with a ^. If *c* is missing *tset* uses control-X.

-I Don't initialize the terminal.

-Q Don't remind user of erase and kill values.

-m*pseudotype test speed*:*type*

Use one or more **-m** options in place of a type argument when you want *tset* to figure out your terminal type for you. *Pseudotype* should be a type that your installation has reserved for a class of "soft" connections, such as **dialup**, **arpanet**, or **plugboard**. A missing pseudotype means "any type.". *Test* and *speed* indicate a class of baud rates. *Test* is = or @ for "equals"; < for "less than"; > for "greater than"; or !=, !@, !<, or !> for negations. A missing speed indication means "all speeds." *Type* is the type to assume if the pseudotype and terminal speeds match. *Type* can begin with a question mark to indicate a user query. Thus,

```
tset - -m 'dialup@300:trs80' -m 'dialup:t0'
```

prints **trs80** if **TERM** is **dialup** and the baud rate is 300; **t0** if **TERM** is **dialup** and the baud rate isn't 300; and the value of **TERM** otherwise.

The login scripts **/etc/profile** and **/etc/cprofile** check for the presence of a file named **/etc/rcopts/TSETX** : if the file is found, these scripts set the **TERM** environment variable as follows:

```
tset - '?dumb'
```

This causes users to be queried about their terminal types when they log in.

FILES

/etc/ttytype	type wired to each port
/etc/termcap	terminal capability database

SEE ALSO

sh(1), stty(1), cprofile(4), profile(4), ttytype(4), termcap(4), environ(5).
S/Series CTIX Administrator's Guide.

DIAGNOSTICS

Nonzero return status if it could not process all options and user input. This is useful to confirm that user entered known terminal type: see *profile*(4) for an example.

NAME

tsioctl - facilitate usage of a tape drive

SYNOPSIS

/usr/local/bin/tsioctl [**-s**] [**-c** cmd] device [arg]

DESCRIPTION

The *tsioctl* command facilitates the use of a tape drive by allowing commands to be issued to a tape controller or by obtaining diagnostic information from a tape controller.

Device is one of the following:

/dev/rmt/c0d0c QIC tape on an S/Series computer other than an S/640.

/dev/rmt/c0d*n*c SCSI tape on an S/640, where *n* is the drive number (0 through 7).

/dev/rmt/c1d*n*c Half-inch tape, where *n* is the drive number (0 through 7).

The following options are recognized by *tsioctl*:

-s Read the status from the tape controller, similar to the read tape status option in the Diagnostic.

-c *cmd* Issue a command to the tape drive, where *cmd* is one of the following:

erase
rewind
retension
skip

Not all commands are supported by all tape drive controllers. In particular, Interphase half-inch tape does not currently support *retension* or *erase*.

arg Skip *arg* number of tape file marks if **-c skip** is specified. (*Arg* defaults to one.)

FILES

/dev/rmt/c0d0c	QIC
/dev/rmt/c0d?c	SCSI QIC on an S/640
/dev/rmt/c1d?c	VME half-inch tape
/dev/rmt/c1d?c	half-inch tape
/dev/rmt/c?d?c	QIC and half-inch

EXAMPLES

The first command below rewinds the tape; the second command skips the first five files on the QIC:

```
tsioctl -c rewind /dev/rmt/c0d0c
tsioctl -c skip /dev/rmt/c0d0c 5
```

The first command below rewinds the tape; the second command skips the first five files on a VME half-inch tape, drive 0:

```
tsioctl -c rewind /dev/rmt/c1d0c
tsioctl -c skip /dev/rmt/c1d0c 5
```

SEE ALSO

qic(7), ipt(7).
S/Series CTIX Administrator's Guide.

NAME

tsort - topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

The *tsort* command produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1).

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

NAME

tty - get the name of the terminal

SYNOPSIS

tty [**-l**] [**-s**]

DESCRIPTION

The *tty* command prints the path name of the user's terminal. The following options are available:

-l Print the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line.

-s Inhibit printing of the terminal path name, allowing one to test just the exit code.

The *tty* command returns one of the following exit codes:

2 Invalid options were specified,

0 Standard input is a terminal,

1 Otherwise.

DIAGNOSTICS

not on an active synchronous line
if the standard input is not a synchronous terminal and **-l** is specified.

not a tty if the standard input is not a terminal and **-s** is not specified.

NAME

uadmin - administrative control

SYNOPSIS

/etc/uadmin cmd fcn

DESCRIPTION

The *uadmin* command provides control for basic administrative functions. This command is tightly coupled to the System Administration procedures and is not intended for general use. It may be invoked only by the super-user.

The arguments *cmd* (command) and *fcn* (function) are converted to integers and passed to the *uadmin* system call.

SEE ALSO

uadmin(2).

NAME

uconf - configure the operating system

SYNOPSIS

uconf [**-v** | **-V**] [**-n** namelist] [**-m** master] [parameter [parameter ...]]

uconf -w [**-n** namelist]

uconf parameter=value [parameter=value ...]

DESCRIPTION

The *uconf* program is used to display tunable parameter values, reconfigure the operating system, or change tunable parameter values in memory of a running CTIX operating system. The *uconf* program examines three files:

- A system *namelist* file (by default, the currently running operating system, **/unix**)
- A *master* device file (by default, **/etc/master**)
- The system file (**/etc/system**)

Defaults for the first two files can be overridden by using the **-n** *namelist* and **-m** *master* options.

The first form of the command prints a summary of current parameter values. A null output indicates that there are no discrepancies in parameter values between the *namelist*, *master*, and system files. The **-v** option specifies verbose output, which displays values of all configurable parameters, truncating the parameter name to ten characters; **-V** specifies verbose output, with full parameter names. The verbose output contains information under the following headings:

MNAME	Parameter name as it appears in the default *master* file (or the file specified with the **-m** option).
SNAME	Parameter name as it appears in the **/etc/system** file.
STRUCT	Structure containing the item.
MASTER	Value specified in the *master* file.
SYSTEM	Value specifed in **/etc/system**.
UNIX	Value in *namelist* file (for example, **/unix**).
KMEM	Value in the running system.
NEW VALUE	Value that would result if *uconf* were invoked with the **-w** option.

Note that if the value for **KMEM** is suffixed with a plus sign (+), a memory write is allowed for that parameter (see the description of the third form of the command, below.

Use the *parameter* argument with the first form to display current information for the specified parameter(s) only. (The display is like that produced by the **-v** option, except that only values for the specified parameters appear.) The *parameter* argument corresponds to the **SNAME** field in the display.

The second form of the command updates the *namelist* file with all values contained in the **/etc/system** file. Note that each time you edit the **/etc/system** file, you must run **uconf -w** to update the operating system image **/unix**. The system must be rebooted for the change to take effect after running **uconf -w**.

The third form of the command changes the value in memory for each *parameter/value* pair specified. Note that this form is valid *only* for parameters for which a memory write is allowed [as designated by a plus sign (+) suffixing the **KMEM** value in the *uconf* display].

ERRORS

Any of the following conditions produces a fatal error:

Cannot open the namelist file.

Cannot open the master or system files.

Cannot read COFF information or bad COFF information in the namelist file.

Cannot allocate memory.

nlist(3C) failed.

Cannot locate parameter in namelist file.

Cannot locate parameters section of the master file.

read (2) or *write* (2) errors.

A warning is issued if the system file cannot be parsed correctly.

FILES

/etc/master
/etc/system
/unix
/dev/kmem

NOTES

To hard-code a loadable driver into the kernel (rather than have it load dynamically), you must remake the kernel.

SEE ALSO

config(1M).
S/Series CTIX Administrator's Guide.

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 appropriate underlining sequence for the terminal in use, as specified by the environment variable **TERM**. The **-t** option overrides the terminal type specified in the environment. The **/etc/termcap** file is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining but is capable of a standout mode, standout mode is used instead. If the terminal can overstrike or handles underlining automatically, *ul* reverts to *cat*(1). If the terminal cannot underline, underlining is ignored.

The **-i** option causes *ul* to indicate underlining by a string of hyphens on the following line. This is useful for looking at the underlining in an *nroff* output stream on a terminal.

SEE ALSO

man(1), nroff(1).

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

umask - set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

The user file-creation mode mask is set to *ooo*, where *ooo* represents three octal digits. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively [see *chmod*(2) and *umask*(2)]. The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file [see *creat*(2)]. For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

umask is recognized and executed by the shell.

The CTIX distribution **/etc/profile** file sets the *umask* to **022**. *umask* can also be included in the user's **.profile** [see *profile*(4)].

SEE ALSO

chmod(1), sh(1), chmod(2), creat(2), umask(2), cprofile(4), profile(4).

NAME

unadv - unadvertise a Remote File Sharing resource

SYNOPSIS

unadv *resource*

DESCRIPTION

The *unadv* command unadvertises a Remote File Sharing (RFS) *resource*, which is the advertised symbolic name of a local directory, by removing it from the advertised information on the domain name server. *unadv* prevents subsequent remote mounts of that resource. It does not affect continued access through existing remote or local mounts.

An administrator at a server can unadvertise only those resources that physically reside on the local machine. A domain administrator can unadvertise any resource in the domain from the primary name server by specifying *resource* name as *domain.resource*. (A domain administrator should unadvertise another hosts resources only to clean up the domain advertise table when that host goes down. Unadvertising another host's resource changes the domain advertise table, but not the host advertise table.)

This command is restricted to the super-user.

ERRORS

If *resource* is not found in the advertised information, an error message is sent to standard error.

SEE ALSO

adv(1M), fumount(1M), nsquery(1M).

NAME

uname - print name of current CTIX system

SYNOPSIS

uname [**-snrvma**]
uname [**-S** system name]

DESCRIPTION

uname prints the current system name of the CTIX system on the standard output file. It is mainly useful to determine which system one is using. The options cause selected information returned by *uname*(2) to be printed:

-s print the system name (default).

-n print the nodename (the nodename is the name by which the system is known to a communications network).

-r print the operating system release.

-v print the operating system version.

-m print the machine hardware name.

-a print all the above information.

-S *system name*
set the system name.

The CTIX distribution files **/etc/profile** and **/etc/cprofile** do a *uname* as part of the login procedure.

SEE ALSO

uname(2), cprofile(4), profile(4).

NAME

unget - undo a previous get of an SCCS file

SYNOPSIS

unget [-r**SID**] [**-s**] [**-n**] files

DESCRIPTION

The *unget* command undoes the effect of a **get -e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

-r*SID* Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The use of this keyletter is necessary only if two or more outstanding *get*s for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

-s Suppresses the printout, on the standard output, of the intended delta's *SID*.

-n Causes the retention of the gotten file which would normally be removed from the current directory.

SEE ALSO

delta(1), get(1), help(1), sact(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

uniq - report repeated lines in a file

SYNOPSIS

uniq [**-udc** [+n] [-n]] [input [output]]

DESCRIPTION

uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **-u** flag is used, just the lines that are not repeated in the original file are output. The **-d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **-u** and **-d** mode outputs.

The **-c** option supersedes **-u** and **-d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

-n The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+n The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

comm(1), sort(1).

NAME

units - conversion program

SYNOPSIS

units

DESCRIPTION

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e-01
```

units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter,
c	speed of light,
e	charge on an electron,
g	acceleration of gravity,
force	same as **g**,
mole	Avogadro's number,
water	pressure head per unit height of water,
au	astronomical unit.

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (for example, **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

FILES

/usr/lib/unittab

NAME

update - provide disk synchronization

SYNOPSIS

update [*s*]

DESCRIPTION

update implements regular synchronization. It executes an infinite loop with two actions:

- A *sleep* for *s* seconds (30 seconds default).
- A *sync* system call. This feature calls for less overhead than the practice of having the *cron* command run *sync*.

Run it only once, in background. The CTIX initialization procedure starts *update* automatically. The initialization routine in **/etc/init.d/sysetup** looks for the presence of a file named **/etc/rcopts/UPDRATE**: if the file exists, the initialization routine sets the frequency between *syncs* to the number of seconds specified in **UPDRATE**. Otherwise, *update* is started with the default frequency (30 seconds).

FILES

/dev/dsk/*	disk interfaces
/etc/init.d/sysetup	initialization routine that includes update command; linked to /etc/rc?.d/S20sysetup

SEE ALSO

rc2(1M), ioctl(2), sleep(3), disk(7).

NAME

usage - retrieve a command description and usage examples

SYNOPSIS

[**help**] **usage** [**-d**] [**-e**] [**-o**] [command_name]

DESCRIPTION

The CTIX system Help Facility command *usage* retrieves information about CTIX system commands. With no argument, *usage* displays a menu screen prompting the user for the name of a command, or allows the user to retrieve a list of commands supported by *usage*. The user may also exit to the shell by typing q (for "quit").

After a command is selected, the user is asked to choose among a description of the command, examples of typical usage of the command, or descriptions of the command's options. Then, based on the user's request, the appropriate information will be printed.

A command name may also be entered at shell level as an argument to *usage*. To receive information on the command's description, examples, or options, the user may use the **-d**, **-e**, or **-o** options respectively. (The default option is **-d**.)

From any screen in the Help Facility, a user may execute a command via the shell [*sh*(1)] by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your *.profile* file [see *profile* (4)]:

```
export SCROLL ; SCROLL=no
```

If you later decide that scrolling is desired, **SCROLL** must be set to **yes**.

Information on each of the Help Facility commands (*starter*, *locate*, *usage*, *glossary*, and *help*) is located on their respective manual pages.

SEE ALSO

glossary(1), help(1), locate(1), sh(1), starter(1), term(5).

WARNINGS

If the shell variable **TERM** [see *sh*(1)] is not set in the user's *.profile* file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5).

NAME

uucheck - check the uucp directories and permissions file

SYNOPSIS

/usr/lib/uucp/uucheck [**-v**] [**-x** debug_level]

DESCRIPTION

uucheck checks for the presence of the *uucp* system required files and directories. Within the *uucp* makefile, it is executed before the installation takes place. It also checks for some obvious errors in the Permissions file (**/usr/lib/uucp/Permissions**). When executed with the **-v** option, it gives a detailed explanation of how the uucp programs will interpret the Permissions file. The **-x** option is used for debugging. *debug-option* is a single digit in the range 1-9; the higher the value, the greater the detail.

Note that *uucheck* can only be used by the super-user or *uucp*.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuscheds
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

uucico(1M), uucp(1C), uusched(1M), uustat(1C), uux(1C).

BUGS

The program does not check file/directory modes or some errors in the Permissions file such as duplicate login or machine name.

NAME

uucico - file transport program for the uucp system

SYNOPSIS

/usr/lib/uucp/uucico [**-r** role_number] [**-x** debug_level] [**-i** interface] [**-d** spool_directory] **-s** system_name

DESCRIPTION

The *uucico* program performs file transport for *uucp* work file transfers. Role numbers for the **-r** are the digit 1 for master mode or 0 for slave mode (default). The **-r** option should be specified as the digit 1 for master mode when *uucico* is started by a program or *cron*. The *uux* and *uucp* programs both queue jobs that are transferred by *uucico*. The *uucico* program is normally started by the scheduler, *uusched*, but it can be started manually; this is done for debugging. For example, *Uutry* starts *uucico* with debugging turned on. A single digit must be used for the **-x** option with higher numbers for more debugging.

The **-i** option defines the interface used with *uucico*. This interface only affects slave mode. Known interfaces are UNIX (default), TLI (basic Transport Layer Interface), and TLIS (Transport Layer Interface with Streams modules, read/write).

The **-d** option specifies the directory (*spool_directory*) that contains the work files to be transferred. The default spool directory is **/usr/spool/uucp**. The **-s** option defines the system (*system_name*) that *uucico* tries to contact. The *system_name* must be defined in the **Systems** file.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Devconfig
/usr/lib/uucp/Sysfiles
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), uucp(1C), uusched(1M), uustat(1C), Uutry(1M). uux(1C).
S/Series CTIX Administrator's Guide.

NAME

uucleanup - uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uucleanup [**-C***time*] [**-W***time*] [**-X***time*] [**-m***string*] [**-o***time*] [**-s***system*]

DESCRIPTION

The *uucleanup* command scans the spool directories for old files and takes appropriate action to remove them in a useful way:

- Inform the requestor of send/receive requests for systems that can not be reached.
- Return mail, which cannot be delivered, to the sender.
- Delete or execute *rnews* for *rnews* type files (depending on where the news originated--locally or remotely).
- Remove all other files.

In addition, there is provision to warn users of requests that have been waiting for a given number of days (default 1). Note that *uucleanup* will process as if all option *times* were specified to the default values unless *time* is specifically set.

The following options are available.

-C*time* Any **C.** files greater or equal to *time* days old will be removed with appropriate information to the requestor. (default 7 days)

-D*time* Any **D.** files greater or equal to *time* days old will be removed. An attempt will be made to deliver mail messages and execute *rnews* when appropriate. *Note: rnews* is public domain software not provided with CTIX. (default 7 days)

-W*time* Any **C.** files equal to *time* days old will cause a mail message to be sent to the requestor warning about the delay in contacting the remote location. The message includes the *JOBID*, and in the case of mail, the mail message. The administrator may include a message line telling whom to call to check the problem (**-m** option). (default 1 day)

-X*time* Any **X.** files greater or equal to *time* days old will be removed. The **D.** files are probably not present (if they were, the **X.** could get executed). But if there are **D.** files, they will be taken care of by D. processing. (default 2 days)

-m*string* This line will be included in the warning message generated by the **-W** option. The default message is

"See your local administrator to locate the problem."

-o*time* Other files whose age is more than *time* days will be deleted. (default 2 days)

-s*system* Execute for *system* spool directory only.

-x*debug_level*
The **-x** debug level is a single digit between 0 and 9; higher numbers give more detailed debugging information.

This program is typically started by the shell script *uudemon.clean*, which should be started by *cron*(1M).

FILES

/usr/lib/uucp	directory with commands used by *uucleanup* internally
/usr/spool/uucp	spool directory

SEE ALSO

cron(1M), uucp(1C), uux(1C).
S/Series CTIX Administrator's Guide.

NAME

uucp, uulog, uuname - UNIX-to-UNIX system copy

SYNOPSIS

uucp [options] source-files destination-file

uulog [options] **-s**system

uulog [options] system

uulog [options] **-f**system

uuname [**-l**] [**-c**]

DESCRIPTION

uucp

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name can be a path name on your machine, or can have the following form:

node!path-name

where *node* is taken from a list of system names that *uucp* knows about. The *node* can also be a list of names such as

node!node!...!node!path-name

in which case an attempt is made to send the file through the specified route, to the destination. See WARNINGS and BUGS below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to foward information (see WARNINGS below for restrictions).

The shell metacharacters **?**, ***** and **[...]** appearing in *path-name* will be expanded on the appropriate system.

Path names can be one of the following:

(1) a full path name;

(2) a path name preceded by *~user* where *user* is a login name on the specified system and is replaced by that user's login directory;

(3) a path name preceded by *~/destination* where *destination* is appended to **/usr/spool/uucppublic**; (NOTE: This destination will be treated as a file name unless more than one file is being transfered by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a slash (/): for example, using **~/dan/** as the destination makes the directory **/usr/spool/uucppublic/dan** if it does not exist and puts the requested file or files in that directory).

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name 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*:

-c Do not copy local file to the spool directory for transfer to the remote machine (default).

-C Force the copy of local files to the spool directory for transfer.

-d Make all necessary directories for the file copy (default).

-f Do not make intermediate directories for the file copy.

-g*grade* *Grade* is a single letter/number; lower ascii sequence characters will cause the job to be transmitted earlier during a particular conversation.

-j Output the job identification ASCII string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.

-m Send mail to the requester when the copy is completed.

-n*user* Notify *user* on the remote system that a file was sent.

-r Do not start the file transfer, just queue the job.

-s*file* Report status of the transfer to *file*. Note that the *file* must be a full path name.

-x*debug_level*
Produce debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

uulog

uulog queries a log file of *uucp* or *uuxqt* transactions in a file **/usr/spool/uucp/.Log/uucico/***system*, or **/usr/spool/uucp/.Log/uuxqt/***system*.

The options cause *uulog* to print logging information:

-s*sys* Print information about file transfer work involving system *sys*.

-f*system* Does a "tail **-f**" of the file transfer log for *system*. (You must hit BREAK to exit this function.) Other options used in conjunction with the above:

-x Look in the *uuxqt* log file for the given system.

-number Indicates that a "tail" command of *number* lines should be executed.

uuname

uuname lists the names of systems known to *uucp*. The **-c** option returns the names of systems known to *cu*. (The two lists are the same, unless your machine is using different *Systems* files for *cu* and *uucp*. See the *Sysfiles* file.) The **-l** option returns the local system name.

FILES

/usr/spool/uucp	spool directories
/usr/spool/uucppublic/*	public directory for receiving and sending (**/usr/spool/uucppublic**)
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail(1), uustat(1C), uux(1C), uuxqt(1M), chmod(2).
S/Series CTIX Administrator's Guide.

WARNINGS

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 path name; 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 path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to ~/).

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.

The forwarding of files through other systems may not be compatible with the previous version of *uucp*. If forwarding is used, all systems in the route must have the same version of *uucp*.

BUGS

Protected files and files that are in protected directories that are owned by the requestor can be sent by *uucp*. However, if the requestor is root, and the directory is not searchable by "other" or the file is not readable by "other," the request will fail.

NAME

uucpd, ouucpd - network uucp servers

SYNOPSIS

/etc/uucpd

/etc/ouucpd

DESCRIPTION

uucpd and *ouucpd* are network servers for CTIX network file transfer using the *uucp* user interface and protocols. *uucpd* is used for file transfers in which the client machine is running a version of *uucp* based on the Honey-Danber or Berkeley implementations. (The Honey-Danber implementation is used on versions of CTIX that are 5.1 and higher.) *uucpd* requires a password for authentication.

ouucpd is used for file transfers in which the client machine is running a Convergent-specific version of *uucp* over TCP/IP prior to the Honey-Danber implementation.

This server is similar to the *rshd* (1M) server, except:

1) The remote socket need not be privileged,

2) The **/usr/lib/uucp/uucico** shell must be invoked, and

3) If a **.rhosts** file does not exist in the home directory of the user specified in the **Systems** file of the client, any host in the **Systems** file on the server will be allowed access.

A network *uucp* connection is indicated with the **TCP**, **UCBTCP**, and **INET** keywords in the configuration file **/usr/lib/uucp/Systems**. The former two connect to *uucpd*, the latter to *ouucpd*. *uucpd* and *ouucpd* are started by the "super-server" *inetd*, and therefore must have entries in *inetd*'s configuration file, **/etc/inetd.conf** [see *inetd*(1M) and *inetd.conf*(4)].

SEE ALSO

inetd(1M), rshd(1), uucp(1C), inetd.conf(4), services(4).

NAME

uugetty - set terminal type, modes, speed, and line discipline

SYNOPSIS

/usr/lib/uucp/uugetty [**-h**] [**-t** timeout] [**-r**] [**-f**] line [speed [type [linedisc]]]
/usr/lib/uucp/uugetty -c file

DESCRIPTION

The *uugetty* program is actually the *getty*(1M) program altered to allow bidirectional use of a line. The *uugetty* program allows users to log in, but if the line is free, *uucico*(1M), *cu*(1C), or *ct*(1C) can use it for dialing out or receiving calls.

The options to *uugetty* are identical to *getty*(1M) options; two additional options are provided: **-r** and **-f**.

The **-r** option causes *uugetty* to wait to read a character before sending the login script, thus preventing two *uugetty*s from looping.

The **-r** option must be used if a *uugetty* is present on both the sending and receiving systems or if the calling system uses an intelligent modem. When the **-r** option is used, several carriage return characters might be required to elicit the login message; human users can handle this slight inconvenience. The expect/send sequence used by *uucico*(1M), however, must be altered in the **/usr/lib/uucp/Systems** file to include the additional carriage return characters and pauses; for example:

"" \r\d\r\d\r\d ogin:--ogin: *systemname* **ssword** *password*

The **-f** option causes input to be ignored for a short time after the line is opened, and again after reading a character if the **-r** flag was specified. It also disables echo until the login prompt is printed, which is useful where modems are being used on systems that print banners and connection information.

When *cu*(1C) or *uucico*(1M) is invoked, it creates a lock file that prevents users from logging in at the device while the process is running.

Note that *uugetty* must be on both the sending and receiving systems (that is, there cannot be a *uugetty* on one and a *getty* on the other).

EXAMPLE

The following **/etc/inittab** entry can be used for an intelligent modem or a direct line:

```
002:2:respawn:/usr/lib/uucp/uugetty -r -t 60 tty002 1200
```

FILES

/etc/gettydefs
/etc/issue
/usr/lib/uucp/Devices
/usr/spool/locks/LCK.*

SEE ALSO

ct(1C), cu(1C), getty(1M), init(1M), login(1), uucico(1M), ioctl(2), gettydefs(4), inittab(4), tty(7).
S/Series CTIX Administrator's Guide.

BUGS

The *ct* command does not work when *uugetty* is used with an intelligent modem such as penril or ventel.

NAME

uusched - the scheduler for the UUCP system

SYNOPSIS

/usr/lib/uucp/uusched [**-x** debug_level] [**-u** debug_level]

DESCRIPTION

uusched is the UUCP scheduler. It is usually started by the demon *uudemon.hour*, which is started by *cron*(1M).

The two options may be used for debugging. **-x** *debug_level* displays debugging messages from *uusched*. **-u** *debug_level* is passed as **-x** *debug_level* to *uucico*(1M). *Debug_level* is a number between 0 and 9; higher numbers give more detailed information.

To invoke *uusched* the following entry must be included in the **/usr/spool/cron/crontabs/root** file:

```
56 * * * * /usr/lib/uucp/uudemon.hour > /dev/null 2>&1
```

When *uusched* is invoked, it checks the **/usr/spool/uucp** directories for files awaiting transfer. If *uusched* encounters a C. (control) file, it invokes *uucico*(1M) to make the transfer.

FILES

/usr/lib/uucp/uudemon.hour
/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/spool/uucp/*
/usr/spool/locks/LCK.*
/usr/spool/uucppublic/*
/usr/spool/cron/crontabs/root

SEE ALSO

cron(1M), uucico(1M), uucp(1C), uustat(1C) uux(1C).
S/Series CTIX Administrator's Guide.

NAME

uustat - uucp status inquiry and job control

SYNOPSIS

uustat [**-a**]

uustat [**-m**]

uustat [**-p**]

uustat [**-q**]

uustat [**-k***jobid*]

uustat [**-r***jobid*]

uustat [**-s***system*] [**-u***user*]

DESCRIPTION

The *uustat* command displays the status of, or cancels, previously specified *uucp* commands, or provides general status on *uucp* connections to other systems. Only one of the following options can be specified with *uustat* per command execution:

-a Display all jobs in queue.

-m Report the status of accessibility of all machines.

-p Execute a ''ps -flp'' for all the process-IDs that are in the lock files.

-q List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in () next to the number of C or X files, it is the age in days of the oldest C./X. file for that system. The Retry field represents the number of hours until the next possible call. The Count is the number of failure attempts. NOTE: for systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute. As an example of the output produced by the **-q** option:

```
eagle    3C  04/07-11:07  NO DEVICES AVAILABLE
mh3bs3 2C  07/07-10:42  SUCCESSFUL
```

The above output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

-k*jobid* Kill the *uucp* request whose job identification is *jobid*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

-r*jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the deamon.

Either or both of the following options can be specified with *uustat*:

-s*sys* Report the status of all *uucp* requests for remote system *sys*.

-u*user* Report the status of all *uucp* requests issued by *user*.

Output for both the -s and -u options has the following format:

```
eaglen0000   4/07-11:01:03 (POLL)
eagleN1bd7   4/07-11:07    S   eagle dan 522 /usr/dan/A
eagleC1bd8   4/07-11:07    S   eagle dan 59 D.3b2al2ce4924
             4/07-11:07    S   eagle dan rmail mike
```

With the above two options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an 'S' or 'R' depending on whether the job is to send or request a file. This is followed by the user-ID of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (*rmail* - the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (for example, D.3b2alce4924) that is created for data files associated with remote executions (*rmail* in this example).

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user.

FILES

/usr/spool/uucp/* spool directories

SEE ALSO

uucp(1C).

NAME

uuto, uupick - public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [options] source-files destination

uupick [**-s** system]

DESCRIPTION

uuto sends *source-files* to *destination*. *uuto* uses the *uucp*(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!*user*

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*). *User* is the login name of someone on the specified system.

Two *options* are available:

-p Copy the source file into the spool directory before transmission.

-m Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. By default this directory is /usr/spool/uucppublic. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by *mail*(1) of the arrival of files.

Uupick accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from *system*: [file *file-name*] [dir *dirname*] ?

Uupick then reads a line from the standard input to determine the disposition of the file:

<new-line>	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir*. If *dir* is not specified as a complete path name (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.

a [*dir*]	Same as **m** except moving all the files sent from *system*.
p	Print the content of the file.
q	Stop.
EOT (control-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

Uupick invoked with the -s*system* option will only search the PUBDIR for files sent from *system*.

FILES

PUBDIR /usr/spool/uucppublic public directory

SEE ALSO

mail(1), uucleanup(1M), uucp(1C), uustat(1C), uux(1C).

WARNINGS

In order to send files that begin with a dot (for example, .profile) the files must by qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

NAME

uux - UNIX-to-UNIX system command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

The *uux* command gathers zero or more files from various systems, executes a command on a specified system, and then sends standard output to a file on a specified system.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names can be one of the following:

(1) a full path name;

(2) a path name preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

As an example, the command:

uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"

gets the *file1* and *file2* files from the **usg**and**pwba** machines, executes a *diff*(1) command, and puts the results in *file.diff* in the local **PUBDIR/dan/** directory.

Any special shell characters, such as **<>;|**, should be quoted, either by quoting the entire *command-string* or by quoting the special characters as individual arguments.

uux attempts to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command:

uux a!cut -f1 b!/usr/file \(c!/usr/file\)

gets **/usr/file** from system **b** and sends it to system **a**, performs a *cut* command on that file, and sends the result of the *cut* command to system **c**.

uux notifies you if the requested command on the remote system was disallowed. This notification can be disabled by use of the **-n** option. The response comes by remote mail from the remote machine.

The following *options* are interpreted by *uux*:

- The standard input to *uux* is made the standard input to the *command-string*.

-a*name* Use *name* as the user identification replacing the initiator user-ID. (Notification will be returned to the user.)

-b Return whatever standard input was provided to the *uux* command if the exit status is non-zero.

-c Do not copy local file to the spool directory for transfer to the remote machine (default).

-C Force the copy of local files to the spool directory for transfer.

-g*grade* *Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.

-j Output the jobid ASCII string on the standard output which is the job identification. This job identification can be used by *uustat* to obtain the status or terminate a job.

-n Do not notify the user if the command fails.

-p Same as -: The standard input to *uux* is made the standard input to the *command-string*.

-r Do not start the file transfer, just queue the job.

-s*file* Report status of the transfer in *file*.

-x*debug_level*
Produce debugging output on the standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

-z Send success notification to the user.

FILES

/usr/spool/uucp/*	spool directories
/usr/lib/uucp/Permissions	remote execution permissions
/usr/lib/uucp/*	other data and programs

SEE ALSO

cut(1), mail(1), uucp(1C), uustat(1C).

NOTES

For security reasons, most installations limit the list of commands executable on behalf of an incoming request from *uux*, permitting only the receipt of mail [see *mail*(1)]. (Remote execution permissions are defined in **/usr/lib/uucp/Permissions**.)

WARNINGS

Only the first command of a shell pipeline can have a *system-name*!. All other commands are executed on the system of the first command.

The use of the shell metacharacter * does not probably do what you want it to do. The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the *uucp* system. All files required for the execution are put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the *uux* request. The following command does *not* work:

uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"

but, the following command *does* work (if *diff* is a permitted command):

uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"

BUGS

Protected files and files that are in protected directories that are owned by the requestor can be sent in commands using *uux*. However, if the requestor is **root**, and the directory is not searchable by **other**, the request fails.

NAME

uuxqt - execute remote command requests

SYNOPSIS

/usr/lib/uucp/uuxqt [**-s** system] [**-x** debug_level]

DESCRIPTION

The *uuxqt* program executes remote job requests from remote systems generated by the use of the *uux* command. (*Mail* uses *uux* for remote mail requests). *Uuxqt* searches the spool directories looking for *X.* files. For each *X.* file, *uuxqt* checks to see if all the required data files are available and accessible, and file commands are permitted for the requesting system. The *Permissions* file is used to validate file accessibility and command execution permission.

Two environment variables are set before the *uuxqt* command is executed:

UU_MACHINE
Is the machine that sent the job (the previous one).

UU_USER
Is the user that sent the job.

These can be used in writing commands that remote systems can execute to provide information, auditing, or restrictions.

The **-x** *debug_level* is a single digit between 0 and 9. Higher numbers give more detailed debugging information.

FILES

/usr/lib/uucp/Permissions
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*

SEE ALSO

mail(1), uucico(1M). uucp(1C), uustat(1C), uux(1C).

NAME

val - validate SCCS file

SYNOPSIS

val -

val [**-s**] [**-r**SID] [**-m**name] [**-y**type] files

DESCRIPTION

The *val* command determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a **-**, and named files.

val has a special argument, **-**, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

-s The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

-r*SID* The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (for example, **r**1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (for example, **r**1.0 or **r**1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.

-mname The argument value *name* is compared with the SCCS %M% keyword in *file*.

-ytype The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, that is, can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, **-y** mismatch;
bit 7 = %M%, **-m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned - a logical **OR** of the codes generated for each command line and file processed.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

val can process up to 50 files on a single command line. Any number above 50 will produce a **core** dump.

NAME

vc - version control

SYNOPSIS

vc [**-a**] [**-t**] [-cchar] [**-s**] [keyword=value ... keyword=value]

DESCRIPTION

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the **-t** keyletter (see below). The default control character is colon (:), except as modified by the **-c** keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with *ed*(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The **-a** keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter Arguments

-a Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in *vc* statements.

-t All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.

-cchar Specifies a control character to be used in place of :.

-s Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements

:dcl keyword[, ..., keyword]

Used to declare keywords. All keywords must be declared.

:asg keyword=value

Used to assign values to keywords. An **asg** statement overrides the assignment for the corresponding keyword on the *vc* command line and all previous **asg**'s for that keyword. Keywords declared, but not assigned values have null values.

:if condition

:

:

:end

Used to skip lines of the standard input. If the condition is true all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.

The syntax of a condition is:

```
<cond>      ::= [ "not" ] <or>
<or>        ::= <and> | <and> "|" <or>
<and>       ::= <exp> | <exp> "&" <and>
<exp>       ::= "(" <or> ")" | <value> <op> <value>
<op>        ::= "=" | "!=" | "<" | ">"
<value>     ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings
not	may only occur immediately after the *if*, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (for example, : 012 > 12 is false). All other operators take strings as arguments (for example, : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

= != > < all of equal precedence
&
|

Parentheses may be used to alter the order of precedence.

Values must be separated from operators or parentheses by at least one blank or tab.

::text

Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the **-a** keyletter.

:on

:off

Turn on or off keyword replacement on all lines.

:ctl char

Change the control character to char.

:msg message

Prints the given message on the diagnostic output.

:err message

Prints the given message followed by:

ERROR: err statement on line ... (915)

on the diagnostic output. *vc* halts execution, and returns an exit code of 1.

SEE ALSO

ed(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

EXIT CODES

0 - normal
1 - any error

NAME

vi - screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [**-t** tag] [**-r** file] [**-L**] [**-l**] [**-wn**] [**-R**] [**-x**] [**-C**]
[**-c** command] file ...

view [**-t** tag] [**-r** file] [**-L**] [-l] [**-wn**] [**-R\f1**] [**-x**] [**-C**] [**-c**
[**-c** command] file ...

vedit [**-t** tag] [**-a** file] [**-L**] [**-l**] [**-wn**] [**-R**] [**-x**] [**-C**]
[**-c** command] file ...

DESCRIPTION

The *vi* (visual) command invokes a display-oriented text editor based on an underlying line editor *ex*(1). It is possible to use the command mode of *ex* from within *vi* and vice-versa. The visual commands are described on this manual page; how to set options (like automatically numbering lines and automatically starting a new output line when you type carriage return) and all *ex*(1) line editor commands are described on the *ex*(1) manual page.

When using *vi*, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

Invocation Options

The following invocation options are interpreted by *vi* (previously documented options are discussed in the NOTES section at the end of this manual page):

-t *tag* — Edit the file containing the *tag* and position the editor at its definition.

-r *file* — Edit *file* after an editor or system crash. (Recovers the version of *file* that was in the buffer when the crash occurred.)

-L — List the name of all files saved as the result of an editor or system crash.

-w*n* — Set the default window size to *n*. This is useful when using the editor over a slow speed line.

-l — LISP mode; indents appropriately for lisp code, the () {} [[and]] commands in *vi* and *open* are modified to have meaning for *lisp* .

-R — Read-only mode; the **readonly** flag is set, preventing accidental overwriting of the file.

-x Encryption option; when used, *vi* simulates the **X** command of *ex*(1) and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of *crypt*(1). The **X** command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **-x** option. *NOTE:* the standard CTIX distribution is the international version, which does not support encryption. (This is described also in the **WARNING** section at the end of this manual page.)

-C Encryption option; same as the **-x** option, except that *vi* simulates the **C** command of *ex*(1). The **C** command is like the **X** command of *ex*(1), except that all text read in is assumed to have been encrypted.

-c *command* Begin editing by executing the specified editor *command* (usually a search or positioning command).

The *file* argument indicates one or more files to be edited.

The *view* invocation is the same as *vi* except that the **readonly** flag is set.

The *vedit* invocation is intended for beginners. It is the same as *vi* except that the **report** flag is set to 1, the **showmode** and **novice** flags are set, and **magic** is turned off. These defaults make it easier to learn how to use *vi*.

vi Modes

Command Normal and initial mode. Other modes return to command mode upon completion. **ESC** (escape; **GO** on Convergent PT/GT terminals) is used to cancel a partial command.

Input Entered by setting any of the following options: **a A i I o O c C s S R .** Arbitrary text may then be entered. Input mode is normally terminated with **ESC** character, or, abnormally, with an interrupt.

Last line Reading input for : / ? or !; terminate by typing a carriage return; an interrupt cancels termination.

COMMAND SUMMARY

The following sequences represent special keys:

ESC	Escape key. **GO** on Convergent Technologies PT/GT terminals.
^*x*	Control key: hold down the **CTRL** key (**CODE** on Convergent Technologies terminals) and press *x*.
CR	**RETURN** or **CARRIAGE RETURN** key.
↑	Circumflex (^). On teletypewriter-style terminals, usually an up arrow (↑).

Sample commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i*text*ESC	insert *text*
cw*new*ESC	change word to *new*
ea*s*ESC	pluralize word (end of word; append s; escape from input state)
x	delete a character
dw	delete a word
dd	delete a line
3dd	delete three lines
u	undo previous change
ZZ	exit *vi*, saving changes
:q!CR	quit, discarding changes
/*text***CR**	search for *text*
^U ^D	scroll up or down
:*cmd***CR**	any *ex* or *ed* command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	**z G |**
scroll amount	**^D ^U**

repeat effect	most of the rest

Interrupting, canceling

ESC	end insert or incomplete cmd
DEL	(delete or rubout) interrupts

File manipulation

ZZ	if file modified, write and exit; otherwise, exit
:wCR	write back changes
:w!CR	forced write, if permission originally not valid
:qCR	quit
:q!CR	quit, discard changes
:e *name***CR**	edit file *name*
:e!CR	reedit, discard changes
:e + *name***CR**	edit, starting at end
:e +*n***CR**	edit starting at line *n*
:e #CR	edit alternate file
:e!#CR	edit alternate file, discard changes
:w *name***CR**	write file *name*
:w! *name***CR**	overwrite file *name*
:shCR	run shell, then return
:! *cmd***CR**	run *cmd*, then return
:nCR	edit next file in arglist
:n *args***CR**	specify new arglist
^G	show current file and line
:ta *tag***CR**	position cursor to *tag*

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a carriage return.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen

^U	scroll up half screen
***n*G**	go to the beginning of the specified line (end default), where *n* is a line number
/pat	next line matching *pat*
?*pat*	previous line matching *pat*
n	repeat last / or ? command
N	reverse last / or ? command
/pat/+n	nth line after *pat*
?*pat***?-***n*	nth line before *pat*
]]	next section/function
[[	previous section/function
(	beginning of sentence
)	end of sentence
{	beginning of paragraph
}	end of paragraph
%	find matching ( ) { or }

Adjusting the screen

^L	clear and redraw window
^R	clear and redraw window if **^L** is → key
zCR	redraw screen with current line at top of window
z-CR	redraw screen with current line at bottom of window
z .CR	redraw screen with current line at center of window
*/pat/***z-CR**	move *pat* line to bottom of window
z*n* **.CR**	use *n*-line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

`` `` ```	move cursor to previous context
` ' `	move cursor to first non-white space in line
m*x*	mark current position with the ASCII lower-case letter *x*
`` `x ``	move cursor to mark *x*
`'x`	move cursor to first non-white space in line marked by *x*

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
-	previous line, at first non-white
CR	return, same as +
↓ or **j**	next line, same column
↑ or **k**	previous line, same column

Character positioning

^	first non white-space character
0	beginning of line
$	end of line
hor →	forward
l or ←	backward
^H	same as ← (backspace)
space	same as → (space bar)
f*x*	find next *x*
F*x*	find previous **x**
t*x*	move to character prior to next *x*
T*x*	move to character following previous *x*
;	repeat last **f F t** or **T**
,	repeat inverse of last **f F t** or **T**

n|	move to column *n*
%	find matching ({) or }

Words, sentences, paragraphs

w	forward a word
b	back a word
e	end of word
)	to next sentence
}	to next paragraph
(	back a sentence
{	back a paragraph
W	forward a blank-delimited word
B	back a blank-delimited word
E	end of a blank-delimited word

Commands for LISP Mode

)	Forward s-expression
}	
(	Back s-expression
{	

Corrections during insert

^H	erase last character (backspace)
^W	erase last word
erase	your erase character, same as **^H** (backspace)
kill	your kill character, erase this line of input
\	quotes your erase and kill characters
ESC	ends insertion, back to command mode
DEL	interrupt, terminates insert mode
^D	backtab one character; reset left margin of *autoindent*
^^D	caret (^) followed by control-d (^D); backtab to beginning of line; do not reset left margin of *autoindent*

0^D	backtab to beginning of line; reset left margin of *autoindent*
^V	quote non-printable character

Insert and replace

a	append after cursor
A	append at end of line
i	insert before cursor
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R*text*ESC	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since **w** moves over a word, **dw** deletes the word that would be moved over. Double the operator, e.g., **dd** to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
!	filter through command
~	toggle change
=	indent for LISP

Miscellaneous Operations

C	change rest of line (**c$**)
D	delete rest of line (**d$**)
s	substitute chars (**cl**)
S	substitute lines (**cc**)
J	join lines
x	delete characters (**dl**)

X	delete characters before cursor (**dh**)
Y	yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked; however, if a buffer is named (using the ASCII lower-case letters **a** - **z**), the text in that buffer is put instead.

3yy	yank three lines
3yl	yank three characters
p	put back text after cursor
P	put back text before cursor
*x***p**	put from buffer *x*
*x***y**	yank to buffer *x*
*x***d**	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
*d***p**	retrieve *d*'th last delete

FILES

/tmp	default directory where temporary work files are placed; it can be changed using the **directory** option (see the *ex(1)* **set** command)
/usr/lib/terminfo/?/*	compiled terminal description database /usr/lib/terminfo/?/* compiled terminal description database

NOTES

Two options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard (see *intro*(1)). A **-r** option that is not followed with an option-argument has been replaced by **-L** and *+command* has been replaced by **-c** *command*.

SEE ALSO

ed(1), edit(1), ex(1).
Programmer's Guide: CTIX Supplement.

WARNING

Due to export restrictions, encryption features are not available in the standard CTIX distribution.

Tampering with entries in **/usr/lib/terminfo/?/*** (for example, changing or removing an entry) can affect programs such as *vi*(1) that expect the entry to be present and correct. In particular, removing the "dumb" terminal may cause unexpected problems.

BUGS

Software tabs using ^**T** work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

NAME

volcopy - make literal copy of file system

SYNOPSIS

/etc/volcopy [options] fsname srcdevice volname1 destdevice volname2

DESCRIPTION

volcopy makes a literal copy of the file system using a blocksize matched to the device. *Options* are:

-a invoke a verification sequence requiring a positive operator response instead of the standard 10 second delay before the copy is made

-s (default) invoke the **DEL if wrong** verification sequence.

-to The output file is a disk section (also called slice or partition), but is to be treated like a tape.

-ti The input file is a disk section, but is to be treated like a tape.

-y Suppress prompts that ask for ''yes/no'' responses and assume ''yes''.

-v If output file is a tape or is to be treated like a tape, do a verification pass to insure that the tape was written correctly.

Other *options* are used only with tapes:

-bpidensity bits-per-inch (that is, **800/1600/6250**)
-feetsize size of reel in feet (that is, **1200/2400**),
-reelnum beginning reel number for a restarted copy,
-buf use double buffered I/O.
-Q Use **-bpi** and **-feet** values appropriate for quarter-inch tape cartridge.

The **-t** option puts tape headers on media other than tape. If **-ti** or **-to** is specified, the ''reel'' capacity is simply the size of the disk section; the ''reel'' is assumed to be on a removable disk, such as a floppy.

For a true tape such as half-inch reel-to-reel or quarter-inch cartridge, capacity is derived from tape length and density.

The program requests length and density information if it is not given on the command line or is not recorded on an input tape label. If the file system is too large to fit on one reel, *volcopy* will prompt for additional reels. Labels of all reels are checked. Tapes may be mounted alternately on two or more drives. If *volcopy* is interrupted, it will ask if the user wants to quit or wants a shell. In the latter case, the user can perform other operations (for example, *labelit*) and return to *volcopy* by exiting the new shell.

The *fsname* argument represents the mounted name (for example: **root**, **u1**, etc.) of the file system being copied.

The *srcdevice* or *destdevice* should be the physical disk section or tape (for example: **/dev/rdsk/c0d0s5**, **/dev/rmt/c0d0**, etc.).

The *volname* is the physical volume name (for example: **pk3**, **t0122**, etc.) and should match the external label sticker. Such label names are limited to six or fewer characters. *Volname* may be - to use the existing volume name.

Srcdevice and *volname1* are the device and volume from which the copy of the file system is being extracted. *Destdevice* and *volname2* are the target device and volume.

Fsname and *volname* are recorded in the last 12 characters of the superblock (**char fsname[6], volname[6];**).

FILES

/etc/log/filesave.log a record of file systems/volumes copied

EXAMPLE

The following command backs up the **root** file system to a tape:

volcopy -a -Q -buf root /dev/rdsk/c0d0s1 d0 /dev/rmt/c0d0 epoch1

SEE ALSO

labelit(1M), fs(4).

WARNINGS

volcopy does not support tape-to-tape copying. Use *dd*(1) for tape-to-tape copying.

BUGS

Only device names beginning **/dev/rmt/** are treated as tapes.

If the **-buf** option is used with the **-v** option, only the writing of the tape (not the verification pass) uses double buffered I/O.

NAME

wait - await completion of process

SYNOPSIS

wait [*n*]

DESCRIPTION

Wait for your background process whose process ID is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

The shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1).

CAVEAT

If you get the error message *cannot fork, too many processes*, try using the *wait* (1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process IDs associated with your login, and to the number the system can keep track of.)

NOTE

The *n* option is not supported under *csh*.

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

If *n* is not an active process ID, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

wall - write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

wall reads its standard input until an end-of-file. It then sends this message to all currently logged-in users preceded by:

Broadcast Message from ...

It is used to warn all users, typically prior to shutting down the system.

The sender must be super-user to override any protections the users may have invoked [see *mesg*(1)].

FILES

/dev/tty*

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**] [names]

DESCRIPTION

wc counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **-lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

NAME

what - identify SCCS files

SYNOPSIS

what [**-s**] files

DESCRIPTION

The *what* command searches the given files for all occurrences of the pattern that *get*(1) substitutes for %Z% [this is **@(#)** at this printing] and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file **f.c** contains:

```
char ident[] = "@(#)identification information ";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command:

```
what f.c f.o a.out
```

will print:

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

what is intended to be used in conjunction with the command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

-s Quit after finding the first occurrence of pattern in each file.

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, otherwise 1. Use *help*(1) for explanations.

BUGS

It is possible that an unintended occurrence of the pattern **@(#)** could be found just by chance, but this causes no harm in nearly all cases.

NAME

who - who is on the system

SYNOPSIS

who [**-uTlHqpdbrtas**] [**-n** *x*] [file]

who am i

who am I

DESCRIPTION

The *who* command can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process ID of the command interpreter (shell) for each current CTIX system user. It examines the **/etc/utmp** file at login time to obtain its information. If *file* is given, that file [which must be in *utmp*(4) format] is examined. Usually, *file* is **/etc/wtmp**, which contains a history of all the logins since the file was last created.

Used with the **am i** or **am I** option, *who* identifies the invoking user.

The general format for output follows:

name [state] line time [idle] [pid] [comment] [exit]

The *name*, *line*, and *time* information is produced by all options except **-q**; the *state* information is produced only by **-T**; the *idle* and *pid* information is produced only by **-u** and **-l**; and the *comment* and *exit* information is produced only by **-a**. The information produced for **-p**, **-d**, and **-r** is explained during the discussion of each option, below.

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

-u Lists only those users currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time the user logged in. The *idle* column contains the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current." If more than 24 hours have elapsed or the line has not been used since boot time, the entry is marked **old**. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process ID of the user's shell. The *comment* is the comment field associated with this line as found in **/etc/inittab** [see *inittab*(4)]. This can contain information about where the terminal is located, the telephone number of the dataset, the type of terminal if hard-wired, and so on.

-T This option is the same as the **-s** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A plus sign (+) appears if the terminal is writable by anyone; a minus sign (-) appears if it is not. Note that **root** can write to all lines with a + or a - in the *state* field. If a bad line is encountered, a **?** is printed.

-l Lists only those lines on which the system is waiting for someone to login. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

-H Prints column headings above the regular output.

-q A quick *who*, displaying only the names and the number of users currently logged on. When used, all other options (except **-n**) are ignored.

-n *x* This option takes a numeric argument, *x*, which specifies the number of users to display per line. Note that *x* must be at least 1. The **-n** option must be used with the **-q** option.

-p Lists any other process that is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *idle* fields have no meaning. The *comment* field shows the ID field of the line from **/etc/inittab** that spawned this process; see *inittab*(4).

-d Displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values [as returned by *wait*(2)], of the dead process. This can be useful in determining why a process terminated.

-b Indicates the time and date of the last reboot.

-r Indicates the current *run-level* of the *init* process. In addition, it produces the process termination status, process ID, and process exit status [see *utmp*(4)] under the *idle*, *pid*, and *comment* headings, respectively.

-t Indicates the last change to the system clock by **root** [determined through use of the *date*(1) command]; see *su*(1).

-a Processes **/etc/utmp** or the named *file* with all options enabled.

-s This default option lists only the *name*, *line*, and *time* fields.

Note to the super-user: after a shutdown to the single-user state, *who* returns a prompt; the reason is that since **/etc/utmp** is updated at login time and there is

no login in single-user state, *who* cannot report accurately on this state. *who am i*, however, returns the correct information.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), init(1M), login(1), mesg(1), su(1M). wait(2), inittab(4), utmp(4).

NAME

whodo - who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

whodo produces formatted and dated output from information in the */etc/utmp* and */etc/ps_data* files.

The display is headed by the date, time and machine name. For each user logged in, device name, user-ID and login time is shown, followed by a list of active processes associated with the user-ID. The list includes the device name, process-ID, CPU minutes and seconds used, and process name.

EXAMPLE

The command:

whodo

produces a display like this:

```
Tue Mar 12 15:48:03 1985
bailey

tty009   mcn     8:51
  tty009  28158   0:29 sh

tty052   bdr    15:23
  tty052  21688   0:05 sh
  tty052  22788   0:01 whodo
  tty052  22017   0:03 vi
  tty052  22549   0:01 sh

sxt/003  lee    10:20
  tty008   6536   0:05 sh
  tty008   6748   0:01 shl
  sxt001    6751   0:01 sh
  sxt002    6761   0:05 sh
```

FILES

/etc/passwd
/etc/ps_data
/etc/utmp

SEE ALSO

ps(1), who(1).

NAME

wm - window management

SYNOPSIS

exec /usr/local/bin/wm [-k][-s][--] [passparam]

DESCRIPTION

The *wm* program is the window manager. It provides services to application programs running under its control and to users using terminals under its control. The window manager can divide the terminal screen into windows that the user can use like separate terminals. Other services include placement, size, scrolling, and synchronization of windows. Note that *wm* requires a Convergent Technologies Programmable Terminal or Graphics Terminal on a cluster line. The window manager must be running for the window management library functions to work.

The window manager is normally executed in place of the user's login shell by the **exec** command in **/etc/profile** or the user's own **.profile**. The window manager then executes the user's shell each time the user splits a window. The SHELL environment variable [normally set by *login*(1M) to **/bin/sh**] provides the full pathname of the initial program run in the windows.

When *wm* starts, the user sees four regions on the screen, going from top to bottom:

message line
: A single line, always at the top of the screen. It holds messages and prompts from application programs.

tag line
: A single line, always above each window, which labels the particular application program or display that is active in the window.

window
: The main display area used by programs. Text input and output to the shell or an application program goes here. The window is a window into a *virtual display*. An application program can use the virtual display as a 28-line screen, regardless of the size of the window. The virtual display is usually larger than the window. Normally the window manager automatically positions the window over the part of the virtual display that contains the cursor. If the user program moves the cursor to a part of the virtual display not in the window, the window manager scrolls the window until the cursor is visible again. The user can also scroll the display (see below).

function key line
A single line, always at the bottom of the screen, that labels the function keys for the currently active window.

The *wm* program accepts user commands activated by the ACTION key; such commands are not seen by the user program. Use the ACTION key like the CODE or SHIFT keys: hold down the ACTION key and press the other key used with it. Holding down the ACTION key changes the function key line to show how ACTION changes the meanings of the function keys.

Valid *wm* user commands follow:

ACTION-F10 (SPLIT)
Split the active window, creating a new window. The new window and its tag line replace the bottom half of the window being split. Any program running in the old window is unaffected. The virtual display of the old window is unchanged, though less of it is visible. The user shell then starts up in the new window.

The new window is *active*; all other windows are *inactive*. Programs running in inactive windows continue to run, but input calls will not return until the user reactivates the window and types something. Keyboard input goes to the active window.

Each window, whether active on inactive, has its own message line, function key line, and cursor, but the terminal only displays them if they belong to the active window. (Application programs can also make the cursor invisible.) If an application program in a inactive window writes to the message line, the message is not visible until you make that window active again.

On Programmable Terminals the active window's tag line is displayed full intensity, with the other tag lines displayed half intensity. On Graphics Terminals the active window's tag line is displayed in bold, with the other tag lines displayed without bold.

When the SPLIT key creates a new window, *wm* automatically provides a program to run in the window. The program is a process group leader; the new process group is controlled by the new window and has terminal file descriptors associated with the new window. The program is a shell unless *wm* was run with the **-k** option. When all processes in the process group die, *wm* automatically closes the window.

The SPLIT key becomes inoperative if the terminal already displays its maximum number of windows or if a user program has disabled window splitting.

ACTION-F9 (BELOW)
: The window below the active window becomes the active window with the old active window becoming inactive. The new active window takes over the message line and the function key line, and its cursor becomes visible.

  ACTION-↓ is the same as ACTION-F9.

ACTION-F8 (ABOVE)
: The window above the active window becomes the active window. ACTION-↑ is the same as ACTION-F8.

ACTION-*n*
: Activate window *n*, where *n* is a number from 1 to 4. A window's number is assigned when it's first created, with a new window getting the lowest unused number. Unless erased by a user program, the window number is displayed on the left end of the tag line.

ACTION-F7 (SWAP ↓)
: The active window and the window below it trade places.

ACTION-F6 (SWAP ↑)
: The active window and the window above it trade places.

ACTION-F5 (SHRINK)
: The active window decreases in size by 1 line. Ignored if the window is already 0 lines long (only the tag line visible).

ACTION-SHIFT-F5
: The active window decreases in size by 4 lines. If the window is already less than 4 lines long, it becomes 0 lines long.

ACTION-CODE-F5
: The active window becomes 0 lines long.

Shrinking the top window increases the size of the window below; shrinking any other window increases the size of the window above.

ACTION-F4 (GROW)
: The active window increases in size by 1 line. Ignored if the other windows are all 0 lines long.

ACTION-SHIFT-F4
: The active window increases in size by 4 lines. If the other windows don't have 4 lines to spare, the active window increases until all other windows are 0 lines long.

ACTION-F3 (MAX)
ACTION-CODE-F4
: The active window increases in size until all other windows are 0 lines long.

Increasing the top window decreases the size of the window below; increasing any other window decreases the size of window below. If the window that would otherwise shrink is already 0 lines long, the next window shrinks. If all the windows below the second or third window are 0 lines long, space comes from the windows above.

ACTION-SCROLL UP
: The active window is scrolled up a line. Ignored if the window already shows the very bottom of the virtual display or if the cursor is on the window's top line.

ACTION-SCROLL DOWN
: The active window is scrolled down a line. Ignored if the window already shows the very top of the virtual display or if the cursor is on the window's bottom line.

The *wm* command understands the following options:

-k Run *keyprompt*(1) in the first window and in manually-created (SPLIT key) windows instead of the shell.

-s Disable the SPLIT key. The user cannot create new windows, but programs running under *wm* still can.

-- End of *wm* options. Subsequent parameters are passed to the shell, *keyprompt*, or the Office Applications Interface, even if they begin with a dash (-). Parameters other than options are passed unchanged to programs executed by *wm*.

The *wm* program uses or sets the following environment parameters:

TERM If already set, *wm* passes it unchanged to its own children. If not already set, *wm* has the terminal identify itself and sets **TERM** to **pt** or **gt** accordingly.

SHELL Name of the shell's executable file. If **-k** and **-c** aren't specified, SHELL is the initial program in the first window and in user-created (SPLIT windows. If **-k** or **-c** is specified, SHELL must still have a useful value, such as */bin/sh*.

TERMPARM If the user's terminal is a Graphics Terminal, *wm* reads the 32 bytes in the terminal's EAPROM, codes them in hexadecimal, and provides its children with those 64 digits in TERMPARM.

SEE ALSO

sh(1).

WARNING

If a program quickly outputs two things at the virtual display's top and bottom, the user can easily miss one of them. This normally is the fault of programs, originally designed for terminals without window features, that use the bottom line as a message line. Use the terminal message line instead.

BUGS

Message sent by *write*(1) appear only in the first window.

Some *csh*(1) features may not work with *wm*.

NAME

write - write to another user

SYNOPSIS

write user [line]

DESCRIPTION

write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from *yourname* (**tty**???) [*date*]...

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n." At that point *write* writes **EOT** on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (for example, **tty000**); otherwise, the first writable instance of the user found in **/etc/utmp** is assumed and the following message posted:

user is logged on more than one place.
You are connected to "*terminal*".
Other locations are:
terminal

Permission to write may be denied or granted by use of the *mesg(1)* command. Writing to others is normally allowed by default. Certain commands, such as *nroff* and *pr*(1) disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write-inhibited terminal.

If the character **!** is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal [that is, (**o**) for "over"] so that the other person knows when to reply. The signal (**oo**) (for "over and out") is suggested when conversation is to be terminated.

FILES

/etc/utmp	to find user
/bin/sh	to execute !

SEE ALSO

mail(1), mesg(1), pr(1), sh(1), who(1).

DIAGNOSTICS

"*user is not logged on*" if the person you are trying to *write* to is not logged on.

"*Permission denied*" if the person you are trying to *write* to denies that permission (with *mesg*).

"*Warning: cannot respond, set mesg -y*" if your terminal is set to *mesg n* and the recipient cannot respond to you.

"*Can no longer write to user*" if the recipient has denied permission (*mesg n*) after you had started writing.

NAME

xargs - construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

command, which may be a shell file, is searched for, using one's **$PATH**. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **-i** flag). Flags **-i**, **-l**, and **-n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (for example, **-l** vs. **-n**), the last flag has precedence. *Flag* values are:

-l*number* *command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **-x** is forced.

-i*replstr* Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line

are thrown away. Constructed arguments may not grow larger than 255 characters, and option **-x** is also forced. {} is assumed for *replstr* if not specified.

-n*number* Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **-x** is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

-t Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

-p Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**-t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.

-x Causes *xargs* to terminate if any argument list would be greater than *size* characters; **-x** is forced by the options **-i** and **-l**. When neither of the options **-i**, **-l**, or **-n** are coded, the total length of all arguments must be within the *size* limit.

-s*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **-s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

-e*eofstr* *eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical **EOF** string if **-e** is not coded. The value **-e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). *xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

xargs will terminate if either it receives a return code of **-1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* [see *sh*(1)] with an appropriate value to avoid accidentally returning with **-1**.

EXAMPLES

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

```
1. ls | xargs -p -i ar r arch
2. ls | xargs -p -l | xargs ar r arch
```

The following will execute *diff*(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

SEE ALSO

sh(1).

NAME

xstr - extract and share strings in C programs

SYNOPSIS

xstr -c *source*

xstr

xstr *source*

DESCRIPTION

xstr creates a version of a C program in which all strings are contained in a single external array, **xstr**. This optimizes the program in two ways:

- Redundant characters are removed from the object file. A string that is identical to a string earlier in the program is eliminated. A string that is a terminal substring of a longer string is also eliminated, but only if *xstr* sees the longer string first.
- The *xstr* array can be made read-only (shared), reducing space and swapping.

Compiling and linking a program with *xstr* requires three changes in the usual procedure:

1. Instead of compiling the source files, pass each source file to *xstr* with the **-c** option (see first synopsis above). This produces a file *x.c* which is compiled in place of *source*.

   X.c contains the same code as *source* but with each string replaced by an expression of the form (**&xstr**[*number*]), where *number* is the appropriate offset in **xstr**. *xstr* also creates or updates the file *strings* in the current directory to include strings encountered in *source*.

   Source can be a **-**, indicating standard input. This is useful when the C preprocessor produces or suppresses strings. The command to use the preprocessor with *xstr* takes the form

   cc -E *source* **| xstr -c -**

2. Run *xstr* without parameters (second synopsis above). *xstr* uses *strings* to create *xs.c*, a file that declares the **xstr** array. Compile **xs.c**.
3. Link the object file compiled from *xs.c* (normally called *xs.o*) together with all the object files produced in step 1.

Strings is only touched when a string is added or removed. Thus *make*(1) can speed things up by making *xs.o* dependent on *strings*.

If a program has a single source file, pass it to *xstr* without the **-c** option (third synopsis above). This creates *x.c* and *xs.c* without touching *strings*.

EXAMPLE

The following makefile uses *xstr* to produce a program from three source files: *main.c*, *uno.c*, and *omega.c*.

```
a.out:  main.o uno.o omega.o xs.o
        cc main.o uno.o omega.o xs.o

xs.o:   strings
        xstr
        cc -c xs.c

.c.o:
        cc -E $*.c | xstr -c -
        cc -c x.c
        mv x.o $*.o
```

FILES

strings	strings found in source
x.c	massaged C source
xs.c	definition of xstr array
/tmp/xs*	Temp file

NAME

yacc - yet another compiler-compiler

SYNOPSIS

yacc [**-vdlt**] grammar

DESCRIPTION

The *yacc* command converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*.

If the **-v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **-d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned ''token codes'' with the user-declared ''token names''. This allows source files other than **y.tab.c** to access the token codes.

If the **-l** flag is given, the code produced in **y.tab.c** will *not* contain any #line constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, when *yacc*'s **-t** option is used, this debugging code will be compiled by default. Independent of whether the **-t** option was used, the runtime debugging code is under the control of **YYDEBUG**, a preprocessor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp,	
yacc.debug, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(1).
UNIX System V Release 3.2 Programmer's Guide.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

CAVEAT

Because file names are fixed, at most one *yacc* process can be active in a given directory at a given time.