ULTRIX

Reference Pages Section 1: Commands M - Z

Order Number: AD-PC0WA-T1
June 1990

Product Version: ULTRIX Version 4.0 or higher

This manual describes commands from M to Z that are available to all ULTRIX users for both RISC and VAX platforms.

digital equipment corporation
maynard, massachusetts

About Reference Pages

The *ULTRIX Reference Pages* describe commands, system calls, routines, file formats, and special files for RISC and VAX platforms.

Sections

The reference pages are divided into eight sections according to topic. Within each section, the reference pages are organized alphabetically by title, except Section 3, which is divided into subsections. Each section and most subsections have an introductory reference page called `intro` that describes the organization and anything unique to that section.

Some reference pages carry a one- to three-letter suffix after the section number, for example, `scan`(1mh). The suffix indicates that there is a "family" of reference pages for that utility or feature. The Section 3 subsections all use suffixes and other sections may also have suffixes.

Following are the sections that make up the *ULTRIX Reference Pages*.

Section 1: Commands

This section describes commands that are available to all ULTRIX users. Section 1 is split between two binders. The first binder contains reference pages for titles that fall between A and L. The second binder contains reference pages for titles that fall between M and Z.

Section 2: System Calls

This section defines system calls (entries into the ULTRIX kernel) that are used by all programmers. The introduction to Section 2, `intro`(2), lists error numbers with brief descriptions of their meanings. The introduction also defines many of the terms used in this section.

Section 3: Routines

This section describes the routines available in ULTRIX libraries. Routines are sometimes referred to as subroutines or functions.

Section 4: Special Files

This section describes special files, related device driver functions, databases, and network support.

Section 5: File Formats

This section describes the format of system files and how the files are used. The files described include assembler and link editor output, system accounting, and file system formats.

Section 6: Games

The reference pages in this section describe the games that are available in the unsupported software subset. The reference pages for games are in the document *Reference Pages for Unsupported Software.*

Section 7: Macro Packages and Conventions

This section contains miscellaneous information, including ASCII character codes, mail addressing formats, text formatting macros, and a description of the root file system.

Section 8: Maintenance

This section describes commands for system operation and maintenance.

Platform Labels

The *ULTRIX Reference Pages* contain entries for both RISC and VAX platforms. Pages that have no platform label beside the title apply to both platforms. Reference pages that apply only to RISC platforms have a "RISC" label beside the title and the VAX-only reference pages that apply only to VAX platforms are likewise labeled with "VAX." If each platform has the same command, system call, routine, file format, or special file, but functions differently on the different platforms, both reference pages are included, with the RISC page first.

Reference Page Format

Each reference page follows the same general format. Common to all reference pages is a title consisting of the name of a command or a descriptive title, followed by a section number; for example, `date`(1). This title is used throughout the documentation set.

The headings in each reference page provide specific information. The standard headings are:

Name	Provides the name of the entry and gives a short description.
Syntax	Describes the command syntax or the routine definition. Section 5 reference pages do not use the Syntax heading.
Description	Provides a detailed description of the entry's features, usage, and syntax variations.
Options	Describes the command-line options.
Restrictions	Describes limitations or restrictions on the use of a command or routine.
Examples	Provides examples of how a command or routine is used.

Return Values	Describes the values returned by a system call or routine. Used in Sections 2 and 3 only.
Diagnostics	Describes diagnostic and error messages that can appear.
Files	Lists related files that are either a part of the command or used during execution.
Environment	Describes the operation of the system call or routine when compiled in the POSIX and SYSTEM V environments. If the environment has no effect on the operation, this heading is not used. Used in Sections 2 and 3 only.
See Also	Lists related reference pages and documents in the ULTRIX documentation set.

Conventions

The following documentation conventions are used in the reference pages.

%	The default user prompt is your system name followed by a right angle bracket. In this manual, a percent sign (%) is used to represent this prompt.
#	A number sign is the default superuser prompt.
user input	This bold typeface is used in interactive examples to indicate typed user input.
`system output`	This typeface is used in text to indicate the exact name of a command, routine, partition, pathname, directory, or file. This typeface is also used in interactive examples to indicate system output and in code examples and other screen displays.
UPPERCASE lowercase	The ULTRIX system differentiates between lowercase and uppercase characters. Literal strings that appear in text, examples, syntax descriptions, and function definitions must be typed exactly as shown.
rlogin	This typeface is used for command names in the Syntax portion of the reference page to indicate that the command is entered exactly as shown. Options for commands are shown in bold wherever they appear.
filename	In examples, syntax descriptions, and routine definitions, italics are used to indicate variable values. In text, italics are used to give references to other documents.
[ ]	In syntax descriptions and routine definitions, brackets indicate items that are optional.
{ | }	In syntax descriptions and routine definitions, braces enclose lists from which one item must be chosen. Vertical bars are used to separate items.

. . .	In syntax descriptions and routine definitions, a horizontal ellipsis indicates that the preceding item can be repeated one or more times.
. . .	A vertical ellipsis indicates that a portion of an example that would normally be present is not shown.
`cat`(1)	Cross-references to the *ULTRIX Reference Pages* include the appropriate section number in parentheses. For example, a reference to `cat`(1) indicates that you can find the material on the `cat` command in Section 1 of the reference pages.

Online Reference Pages

The ULTRIX reference pages are available online if installed by your system administrator. The `man` command is used to display the reference pages as follows:

To display the `ls`(1) reference page:

```
% man ls
```

To display the `passwd`(1) reference page:

```
% man passwd
```

To display the `passwd`(5) reference page:

```
% man 5 passwd
```

To display the Name lines of all reference pages that contain the word "passwd":

```
% man -k passwd
```

To display the introductory reference page for the family of 3xti reference pages:

```
% man 3xti intro
```

Users on ULTRIX workstations can display the reference pages using the unsupported `xman` utility if installed. See the `xman`(1X) reference page for details.

Reference Pages for Unsupported Software

The reference pages for the optionally installed, unsupported ULTRIX software are in the document *Reference Pages for Unsupported Software*.

Name

m4 – macro processor

Syntax

m4 [*options*] [*files*]

Description

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

Options

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 4,096.
–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, these flags 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 a defined macro name is not followed by a left parenthesis, it is deemed to have no arguments.

Leading unquoted blanks, tabs, and new lines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore (_), where the first character is not a digit.

Left and right single quotes (` ´) are used to quote strings. The value of a quoted string is the string stripped of the quotes.

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

The m4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string. **$#** is replaced by the number of arguments; **$*** is replaced by a list of all the arguments separated by commas; **$@** is like **$***, but each argument is quoted (with the current quotes).

undefine removes the definition of the macro named in its argument.

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

pushdef like *define*, but saves any previous definition.

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

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

changequote Change quote characters to the first and second arguments. The *changequote* without arguments restores the original values (that is, ` ´).

changecom change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.

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

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

divnum	returns the value of the current output stream.
dnl	reads and discards characters up to and including the next new line.
ifelse	has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.
incr	returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
decr	returns the value of its argument decremented by 1.
eval	evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, –, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
len	returns the number of characters in its argument.
index	returns the position in its first argument where the second argument begins (zero origin), or –1 if the second argument does not occur.
substr	returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
shift	is an unimplemented macro. Using shift generates an error message.
translit	transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	returns the contents of the file named in the argument.
sinclude	is identical to *include,* except that it says nothing if the file is inaccessible.
syscmd	executes the UNIX command given in the first argument. No value is returned.
sysval	is the return code from the last call to *syscmd.*
maketemp	fills in a string of XXXXX in its argument with the current process id.
m4exit	causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.

m4wrap	argument 1 will be pushed back 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, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
traceoff	turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

See Also

"The M4 Macro Processor," *ULTRIX Supplementary Documents* Vol. II:Programmer

Name

machine – return architecture type of machine

Syntax

machine

Description

The `machine` command prints on the standard output the architecture of the machine. Legal values are either `mips` or `vax` depending upon your hardware. The `machine` command is used within shell procedures to tailor the results to a specific architecture.

The exit value of `machine` is always zero.

mail(1)

Name

mail – send or read mail

Syntax

mail [**–v**] [**–i**] [**–n**] [**–e**] [**–s** *subject*] [*user...*]
mail [**–v**] [**–i**] [**–n**] **–f** [*name*]
mail [**–v**] [**–i**] [**–n**] **–u** *user*
mail nodename::username (If DECnet is installed.)

Description

The `mail` utility is an intelligent mail processing system which has a command syntax similar to `ed`. However, in `mail` lines are replaced by messages.

If DECnet is installed on your system, you can also send and receive mail from other DECnet users. See `mailaddr`(7) for information on DECnet addressing.

Sending mail. To send a message to one or more persons, type `mail` and the names of the people receiving your mail. Press the RETURN key. Note that if you use other arguments, the names of the recipients should always be the last element on the command line. For example,

```
mail -v -s "mail message" users
```

If you do not specify a subject on the command line, you are prompted for a subject. After entering a subject, and pressing the RETURN key, type your message. To send the message, type a period (.) or CTRL D at the beginning of a new line.

You can use tilde (~) escape sequences to perform special functions when composing mail messages. See the list of options for more on tilde escape sequences.

Reading mail. In normal usage `mail` is given no arguments and checks your mail out of the mail directory. Then it prints out a one line header of each message there. The current message is initially the first message and is numbered 1. It can be displayed using the `print` command.

The `-e` option causes mail not to be printed. Instead, an exit value is returned. For the exit status, see RETURN VALUES. You can move among the messages by typing a plus sign (+) followed by a number to move forward that many messages, or a minus sign (–) followed by a number to move backward that many messages.

Disposing of mail. After reading a message you can delete (d) it or reply (r) to it. Deleted messages can be undeleted, however, in one of two ways: you can use the undelete (u) command and the number of the message, or you can end the mail session with the exit (x) command. Note that if you end a session with the quit (q) command, you cannot retrieve deleted messages.

Specifying messages. Commands such as `print` and `delete` can be given a list of message numbers as arguments. Thus, the command

```
delete 1 2
```

deletes messages 1 and 2, while the command

```
delete 1-5
```

deletes messages 1 through 5. The asterisk (*) addresses all messages, and the dollar sign ($) addresses the last message. For example, the `top` command, which prints

the first few lines of a message, can be used in the following manner to print the first few lines of all messages:

```
top *
```

Replying to or originating mail. Use the `reply` command to respond to a message.

Ending a mail processing session. End a `mail` session with the `quit` (q) command. Unless they were deleted, messages that you have read go to your *mbox* file. Unread messages go back to the mail directory. The –f option causes `mail` to read in the contents of your *mbox* (or the specified file) for processing. When you `quit`, the mail utility writes undeleted messages back to this file. The –u flag is a short way of specifying: `mail -f /usr/spool/mail/user`.

Personal and systemwide distribution lists. You can create a personal distribution list that directs mail to a group of people. Such lists can be defined by placing a line similar to the following in the .mailrc file in your home directory:

```
alias cohorts bill ozalp jkf mark kridle@ucbcory
```

Cohorts is the name of the distribution list that consists of the following users: bill, ozalp, jkf, mark, and kridle@ucbcory. A list of current aliases can be displayed with the `alias (a)` command in `mail`.

System wide distribution lists can be created by editing `/usr/lib/aliases`. The syntax of system wide lists differs from that of personally defined aliases.

Personal aliases are expanded in mail you send. When a recipient on a personally defined mailing list uses the `reply (r)` option, the entire mailing list receives the response automatically. System wide *aliases* are not expanded when the mail is sent, but any reply returned to the machine will have the system-wide alias expanded as all mail goes through `sendmail`.

Forwarding is also a form of aliasing. A `.forward` file can be set up in a user's home directory. Mail for that user is then redirected to the list of addresses in the `.forward` file. See `aliases`(5) and `sendmail`(8) for more information.

Network mail (ARPA, UUCP, Berknet, DECnet) See `mailaddr`(7) for a description of network addresses.

Options

–e Causes `mail` not to be printed. Instead, an exit value is returned.

–f Causes `mail` to read in the contents of your *mbox* file (or another file you specify) for processing.

–i Causes tty interrupt signals to be ignored. This is useful when using `mail` on noisy phone lines.

–n Inhibits the reading of /usr/lib/Mail.rc.

–s Specifies a subject on the command line. Note that only the first argument after the –s flag is used as a subject and that you must enclose subjects containing spaces in quotes.

–u Specifies a short hand for expressing the following:

```
mail -f /usr/spool/mail/user
```

–v Prints the mail message. The details of delivery are displayed on the user's terminal.

The following options can be set in the .mailrc file to alter the behavior of the `mail` command.

Each command is typed on a line by itself and may take arguments following the command word and the command abbreviation. For commands that take message lists as arguments, if no message list is given, then the next message forward which satisfies the command's requirements is used. If there are no messages forward of the current message, the search proceeds backwards. If there are no good messages at all, `mail` cancels the command, displaying the message: No applicable messages.

– Prints out the previous message. If given a numeric argument *n*, prints *n*-th previous message.

? Prints a brief summary of commands.

! Executes the ULTRIX shell command which follows.

alias (a) Prints out all currently-defined aliases, if given without arguments. With one argument, prints out that alias. With more than one argument, creates a new or changes an old alias. These aliases are in effect for the current mail session only.

alternates (alt)
Informs `mail` that you have several valid addresses. The `alternates` command is useful if you have accounts on more than one machine. When you `reply` to messages, `mail` does not send a copy of the message to any of the addresses listed on the *alternates* list. If the `alternates` command is given with no argument, the current set of alternate names is displayed.

chdir (ch) Changes the user's working directory to that specified. If no directory is given, then the `chdir` command changes to the user's login directory.

copy (co) Takes a message list and file name and appends each message to the end of the file. The `copy` command functions in the same way as the `save` command, except that it does not mark the messages that you copy for deletion when you quit.

delete (d) Takes a list of messages as argument and marks them all as deleted. Deleted messages are not saved in *mbox*, nor are they available for most other commands.

dp (or **dt**) Deletes the current message and prints the next message. If there is no next message, `mail` returns a message: at EOF.

edit (e) Takes a list of messages and points the text editor at each one in turn. On return from the editor, the message is read back in.

exit (ex or **x)** Returns to the Shell without modifying the user's system mailbox, *mbox* file, or edit file in `-f`.

file (fi) Switches to a new mail file or folder. If no arguments are given, it tells you which file you are currently reading. If you give it an argument, it writes out changes (such as deletions) you have made in the current file and reads in the new file. Some special conventions are

recognized for the name. A pound sign (#) indicates the previous file, a percent sign (%) indicates your systemb mailbox, %user indicates the user's system mailbox, an ampersand (&) indicates your ~/mbox file, and +folder indicates a file in your folder directory.

folders List the names of the folders in your folder directory.

folder (fo) Switches to a new mail file or folder. The `folder` command functions in the same way as the `file` command.

from (f) Takes a list of messages and prints their message headers in the order that they appear in the mail directory, not in the order given in the list.

headers (h) Lists the current range of headers, which is an 18 message group. If a plus sign (+) is given as an argument, then the next message group is printed. If a minus sign (–) is given as an argument, the previous message group is printed.

help Prints a brief summary of commands. Synonymous with ?.

hold (ho, also preserve)
Takes a message list and marks each message in it to be saved in the user's system mailbox instead of in *mbox*. The `hold` command does not override the **delete** command.

ignore Adds the list of header fields named to the *ignored list*. Header fields in the ignore list are not printed on your terminal when you print a message. This command is frequently used to suppress certain machine-generated header fields. The `type` and `print` commands are used to print a message in its entirety, including ignored fields. If `ignore` is executed with no arguments, it lists the current set of ignored fields.

`mail`**(m)** Takes login names and distribution group names as arguments and sends mail to those people.

mbox Indicates that a list of messages should be sent to *mbox* in your home directory when you quit. This is the default action for messages if you did *not* set the *hold* option.

next (n, + or CR)
Goes to the next message in sequence and types it. With an argument list, it types the next matching message.

preserve (pre)
Takes a message list and marks each message in it to be saved in the user's system mailbox instead of in *mbox* . Synonymous with the `hold` command.

print (p) Takes a message list and types out each message on the user's terminal, without printing any specified ignored fields.

Print (P) Prints a message in its entirety, including specified ignored fields.

quit (q) Terminates the session. All undeleted, unsaved messages are saved in the user's *mbox* file in his login directory; all messages marked with `hold` or `preserve` or that were never referenced are saved in his system mailbox; and all other messages are removed from his system mailbox. If new mail arrives during the session, the user receives the

message: You have new mail. If given while editing a mailbox file with the `-f` flag, then the edit file is rewritten. A return to the Shell is effected, unless the rewrite of the edit file fails, in which case the user can escape with the `exit` command.

reply (r) Takes a message list and sends mail to the sender and all recipients of the specified message. The default message must not be deleted.

Reply (R) Replies to originator of the message. Does not reply to other recipients of the original message.

respond Takes a message list and sends mail to the sender and all recipients of the specified message. Synonymous with `reply`.

save (s) Takes a message list and a file name and appends each message to the end of the file. The messages are saved in the order in which they appear in the mail directory, not in the order given in the message list. The filename, which is enclosed in quotes, followed by the line count and character count, is displayed on the user's terminal.

set (se) Prints all variable values when no arguments are given. Otherwise, the `set` command sets the specified option. Arguments either take the form

```
option=value
```

or

```
option
```

shell (sh) Invokes an interactive version of the shell.

size Takes a message list and prints out the size (in characters) of each message. The size of the messages are printed in the order that they appear in the mail directory, not in the order given in the list.

source (so) Reads `mail` commands from a file.

top Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the variable `toplines` and defaults to five.

type (t) Takes a message list and types out each message on the user's terminal, without printing any specified ignored fields. Synonymous with `print`.

type (T) Prints a message in its entirety, including specified ignored fields. Synonymous with `print`.

unalias Takes a list of names defined by `alias` commands and cancels the list of users. The group names no longer have any significance.

undelete (u) Takes a message list and marks each one as *not* being deleted.

unset Takes a list of option names and discards their remembered values; the inverse of `set`.

visual (v) Takes a message list and invokes the display editor on each message.

write (w) Takes a message list and a file name and appends each message to the end of the file. Synonymous with `save`.

xit (x) Returns to the Shell without modifying the user's system mailbox, *mbox* , or edit file in `-f`. Synonymous with `exit`.

z Presents message headers in windowfulls as described under the `headers` command. You can move forward to the next window with the **z** command. Also, you can move to the previous window by using **z–**.

The following is a summary of the tilde escape functions that you can use when composing mail messages. Note that you can only invoke these functions from within the body of a mail message and that the sequences are only executed if they are placed at the beginning of lines.

~!command Executes the indicated shell command, then returns to the message.

~? Prints a brief summary of tilde commands.

~: Executes the mail commands. (For example, the command `~:10` prints out message number 10 while `~:-` prints out the previous message.

~c name ... Adds the given names to the list of carbon copy recipients.

~d Reads the file named dead letter from your home directory into the message.

~e Invokes the text editor on the message you are typing. After the editing session is finished, you may continue appending text to the message.

~f messages Reads the named messages into the message being sent. If no messages are specified, reads in the current message.

~h Edits the message header fields by typing each one in turn and allowing the user to append text to the end or to modify the field by using the current terminal erase and kill characters.

~m messages Reads the named messages into the message being sent, shifted one tab space to the right. If no messages are specified, reads the current message.

~p Prints the message on your terminal, prefaced by the message header fields.

~q Aborts the message being sent, copying the message to dead.letter in your home directory if the **save** option is set.

~r filename Reads the named file into the message.

~s string Causes the named string to become the current subject field.

~t name ... Adds the given names to the direct recipient list.

~v Invokes an alternate editor (defined by the VISUAL option) on the message. Usually, the alternate editor is a screen editor. After you quit the editor, you can resume appending text to the end of your message.

~w filename Writes the message onto the named file.

~|command Pipes the message through the command as a filter. If the command

gives no output or terminates abnormally, retains the original text of the message. The command fmt(1) is often used as *command* to rejustify the message.

~~string — Inserts the string of text in the message prefaced by a single tilde (~). If you have changed the escape character, then you should double that character in order to send it.

Options are controlled via the `set` and `unset` commands. Options may be either binary or string. If they are binary you should see whether or not they are set; if they are string it is the actual value that is of interest.

The binary options include the following:

append — Causes messages saved in *mbox* to be appended rather than prepended. (This is set in /usr/lib/Mail.rc on version 7 systems.)

ask — Causes *mail* to prompt you for the subject of each message you send. If you simply respond with a new line, no subject field is sent.

askcc — Asks you at the end of each message whether you want to send a carbon copy of the message to additional recipients. Responding with a new line indicates your satisfaction with the current list.

autoprint — Causes the `delete` command to behave like `dp` – thus, after deleting a message, the next one is typed automatically.

debug — Causes `mail` to output information useful for debugging `mail`. Setting the binary option *debug* is the same as specifying `-d` on the command line.

dot — Causes `mail` to interpret a period alone on a line as the terminator of a message you are sending.

hold — Holds messages in the system mailbox by default.

ignore — Causes interrupt signals from your terminal to be ignored and echoed as at signs (@).

ignoreeof — Causes `mail` to refuse to accept a control-d as the end of a message.

msgprompt — Prompts you for the message text and indicates how to terminate the message.

metoo — Includes the sender in the distribution group receiving a mail message.

nosave — Prevents `mail` from copying aborted messages into the dead.letter file in your home directory.

quiet — Suppresses the printing of the version when first invoked.

verbose — Displays the details of each message's delivery on the user's terminal. Setting the `verbose` option is the same as typing **–v** on the command line.

The string options include the following:

EDITOR — Pathname of the text editor to use in the `edit` command and ~e escape. If not defined, then a default editor is used.

SHELL	Pathname of the shell to use in the ! command and the ~! escape. A default shell is used if this option is not defined.
VISUAL	Pathname of the text editor to use in the `visual` command and ~v escape.
crt	Threshold to determine how long a message must be before `more` is used to read it.
escape	The first character of this option gives the character to use in the place of tilde (~) to denote escapes, if defined.
folder	Directory name to use for storing folders of messages. If this name begins with a backslash (/) `mail` considers it an absolute pathname; otherwise, the folder directory is found relative to your home directory.
record	Pathname of the file used to record all outgoing mail. If it is not defined, then outgoing mail is not so saved.
toplines	The number of lines of a message that is printed out with the **top** command; normally, the first five lines are printed.

Return Values

If mail is invoked with the **–e** option, the following exit values are returned:

0 the user has mail
1 the user has no mail

Files

/usr/spool/mail/*	mail directory
~/mbox	your read mail
~/.mailrc	file giving initial mail commands
/tmp/R#	temporary for editor escape
/usr/lib/Mail.help*	help files
/usr/lib/Mail.rc	system initialization file
Message*	temporary for editing messages

See Also

binmail(1), fmt(1), newaliases(1), aliases(5), mailaddr(7), sendmail(8)

make(1)

Name

make, s5make – maintain, update, and regenerate groups of programs

Syntax

make [**-f** *makefile*] [*options*] [*names*]

s5make [**-f** *makefile*] [*options*] [*names*]

Description

This is the SYSTEM V version of the `make` command with some Berkeley compatibility added.

Options

–b	Compatibility mode for old makefiles.
–d	Debug mode. Displays detailed information on files and times examined.
–e	Causes environment variables to override assignments within makefiles.
–f *makefile*	Uses the specified description file name. A file name of – denotes the standard input. The contents of the file specified as *makefile* override the built-in rules.
–i	Ignores error codes returned by invoked commands. This mode is entered if the special target name .IGNORE appears in the description file.
–k	Stops work on the current entry, but continues on other branches that do not depend on that entry.
–m	Displays a memory map showing text, data, and the stack. Does not operate on systems without the `getu` system call.
–n	No execute mode. Displays commands, but does not execute them. Even lines beginning with an at sign (@) are printed.
–p	Displays the complete set of macro definitions and target descriptions.
–q	Question mode. Returns a zero or nonzero status code depending on whether the target file is or is not up to date.
–r	Does not use the built-in rules.
–s	Silent mode. Suppresses the display of command lines before executing. This mode is also entered if the special target name .SILENT appears in the description file.
–S	Abandon work on the current entry if it fails; the opposite of the **-k** option. If both options are specified, the last one specified on the command line is used.
–t	Touches target files (causing them to be up to date) rather than

issuing usual commands.

Special Names

.DEFAULT	If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name .DEFAULT are used if it exists.
.PRECIOUS	Dependents of this target are not removed when quit or interrupt is hit.
.SILENT	Same effect as the **–s** option.
.IGNORE	Same effect as the **–i** option.
.SUFFIXES	Dependencies of the .SUFFIXES special target are added to the table of known suffixes.

Discussion

The `make` program executes commands in *makefile* to update one or more target *names* . The *name* argument is typically a program. If no **–f** option is present, **makefile**, **Makefile**, **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is –, the standard input is taken. You can specify more than one **–f** *makefile* argument.

The `make` program 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.

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

```
echo a\
b
```

These entries produce the following:

```
ab
```

This output is exactly the same as what would have been produced by the shell.

Number sign (#) 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 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 nonzero 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 stops 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. The difference between the old version of `make` and this version is that this version requires all dependency lines to have a (possibly null or implicit) command associated with them. The previous version of `make` assumed, if no command was specified explicitly, that the command was null.

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

Environment

The environment is always read by `make`. All variables are assumed to be macro definitions and processed as such. The **–e** option causes the environment to override the macro assignments in a makefile.

The `make` command operates in three compatibility modes. The type of mode is determined by value of the PROG_ENV environment variable and by the way that `make` is executed. The PROG_ENV variable has three valid values:

- BSD
- POSIX
- SYSTEM_FIVE

In BSD mode, `make` executes with Berkeley compatibility. This means that `/bin/sh` is always used as the command interpreter regardless of the value of SHELL, and the commands are echoed to standard out without a prefixed <tab>.

In POSIX mode, `make` executes with POSIX compatibility, such that the SHELL environment variable is always ignored, SHELL is always overridden by MAKESHELL, the shell is always used to execute commands, and commands are echoed to standard out with a prefixed <tab>.

SYSTEM_FIVE mode causes `make` to run with SYSTEM V compatibility such that SHELL is used to execute commands and commands are echoed to standard out with a prefixed <tab>.

For all modes, SHELL has a default value of `/bin/sh`. When `make` is executed with the command name `s5make`, it always executes in SYSTEM_FIVE mode and ignores the environment variable PROG_ENV.

The MAKEFLAGS environment variable is processed by `make` as containing any legal input option (except **–f, –p,** and **–d**) 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.

Macros

Macros can be defined in four different ways. Some macros are defined by default by `make` internally. All environment variables are assumed to be macro definitions and macros can be defined in the makefile as well as on the `make` command line. By default, the internal default macros are overridden by environment variables, macros defined in the makefile override environment variables and macros defined on the command line override macros defined in the makefile. The **-e** option changes this such that environment variables override macros defined in the makefile.

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*. The occurrence of *subst1* must be a suffix at the end of the word *string1*. 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.**

The MACHINE macro is defined by `make` to allow for machine independent makefiles. The legal values are: *vax* or *mips*.

Internal Macros

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

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

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

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

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

or:

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

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

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

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

Suffixes

Certain names (for instance, those ending with **.o**) have prerequisites such as **.c**, **.s**, which can be inferred. 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 which 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~ .sh .sh~ .c.o .c~.o .c~.c .s.o .s~.o .y.o
.y~.o .l.o .l~.o .y.c .y~.c .l.c .c.a .c~.a .s~.a .h~.h
```

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 out the rules compiled into `make` in a form suitable for recompilation, the following command is used from `/bin/sh`:

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

The only peculiarity in this output is the (null) string which `printf`(3s) prints when handed a null string.

A tilde in the above rules refers to an SCCS file. 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 the `make` 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 .y .l .s
```

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

Inference Rules

The first example can be done more briefly.

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

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

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

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

Libraries

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

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
   @echo lib is now up-to-date
.c.a:
   $(CC) -c $(CFLAGS1) $<
   ar rv $@ $*.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 rv lib $?
   rm $?
   @echo lib is now up-to-date
.c.a:;
```

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

type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

Restrictions

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty. File names with the characters = **:** **@** do not work. Commands that are directly executed by the shell, notably cd(1), are ineffectual across new-lines 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.

Files

[Mm]akefile and s.[Mm]akefile

See Also

cc(1), cd(1), lex(1), sh(1), yacc(1), s5make(1)

Name

man – displays manual pages online

Syntax

man –k *keyword...*
man –f *page_title...*
man [*options*] [*–roff_options*]... [*section*] *page_title...*
man [*options*] [*section page_title...*]...

/usr/bin/man [*option*] [*section*] *page_title...*
/usr/bin/man [*options*] [*section page_title...*]...

Description

There are two `man` commands: `/usr/ucb/man`, and `/usr/bin/man`. For most users, `/usr/ucb/man` is the default `man` command. The command

```
which man
```

shows you which `man` command is the default. The recommended default is `/usr/ucb/man`.

Both the `man` commands provide online displays of specified reference pages.

The /usr/ucb/man Command.

The basic function of this command is to provide online displays of reference pages. You can use *options,* however, to direct the `man` command to display one line summaries of reference pages which contain specific keywords, to display one line summaries of specific reference pages, to use special formatting options when preparing the reference page for display or printing, and to search alternate reference page directories for specified reference pages.

If an option is not used, the `man` command formats and displays one or more specified reference pages. If there are multiple reference pages which match a specified name, only the first matching reference page is displayed. If there are multiple matches in a section for a specified name, the matching page in the first alphabetically occurring subsection is displayed.

If you specify the `man` command with a *section* argument, the `man` command looks in that section of the reference pages for the specified page titles. A *section* consists of a number in the range 0 to 9, optionally followed by an alphanumeric subsection, or *section* can be the name 'local', 'new', 'old', or 'public'. Numbers 0 and 9 are non-standard. If a `section` is omitted, the `man` command searches all sections of the reference pages. The `man` command displays commands (both standard and local) over subroutines in system libraries, and displays the first occurrence it finds, if any. If Section 1 is specified, the sections n, l, and o are also searched, if they exist.

The *section* and *page_title...* arguments can be paired, so that multiple pages can be searched for in a section, and multiple sections can be searched for a page or pages.

All displays are directed to standard out, unless redirected, or unless the `-t` option is specified. If the standard output is a teletype device, the `man` command completes the following: pipes the output through the `cat`(1) command using the option –s to eliminate unnecessary blank lines and invokes the `page`(1) command using the

option –s to display a screen at a time.

If a specified reference page is not already formatted, but the source file exists, the man command preprocesses the file through the tbl(1) command. The command next pipes the output to the nroff(1) command, or to the troff command if the -t option was specified, using the man(7) macros package. If the tbl output was directed to the nroff command, the output is then piped through the col(1) postprocessor, then directed to standard out. If the appropriate /usr/man/cat? directory exists, the formatted display is saved there.

The /usr/bin/man Command

The /usr/bin/man command performs the same basic function as the /usr/ucb/man command, that is, formats and displays or prints specified reference pages. It does not provide all the functions of /usr/ucb/man, and there are some differences in common functions.

The /usr/bin/man command searches for the specified reference pages, and formats and displays all reference pages matching the specified names. If no *section* is specified, all sections 1 through 8 are searched. In the case of multiple matches, the display order is in numeric section order, and ASCII subsection order within a section.

All displays are directed to standard out, unless redirected, or unless an option is used which requests processing through the troff command. If a troff option was not specified, then the standard output is to a teletype device. If the standard output is a teletype device, the /usr/bin/man pipes the output through the col(1) postprocessor, then directs the output to standard out.

The /usr/bin/man command does not use preformatted files. It searches only the /usr/man/man[1-8] directories for source files.

If a specified file exists, it is always preprocessed through the tbl(1) command.

If an option is not used, the /usr/bin/man command formats and displays specified reference pages using the nroff command.

If multiple options are specified, only the last one is executed, except that multiple *-roff_options are accepted and executed.*

Options

/usr/ucb/man Options

The following options are recognized only by the /usr/ucb/man command. Note that the options - and -k do not have the same functionality as the corresponding /usr/bin/man options.

–	Squeeze multiple blank lines from output.
–f	Display one line summaries of each page title specified on the command line.
–k	Display one line summaries of each reference page that contains the specified keyword or keywords.
–P *manpath*	Search the specified *manpath* directory instead of /usr/man.
–s	Remove unnecessary blank lines.

–t Phototypesets the output through the `troff` command.

This option requires the installation of the `troff` command, which is unsupported. When the **–t** option is specified, the `troff` output is directed, by `lpr`'s **–t** option, to the printer or typesetter specified by the PRINTER environment variable. PRINTER must be set to a printer which is capable of handling `troff` output files. The default is the lp printer (see `lpr`(1) description of the **–t** option for more information).

/usr/bin/man Options

The following options are recognized by the `/usr/bin/man` command. Note that the options – and `-k` do not have the same functionality as the corresponding `/usr/ucb/man` options.

–roff_options Inserts the specified *roff_option* in front of the **–man** option when the appropriate *roff text formatter is called (the other options determine which *roff formatter is called). Multiple *roff_options* can be specified. If a null value is specified, the results are unpredictable.

–e | –et | –te Preproceses the display with the `eqn` command, then performs the same steps as the `-t` option.

This option requires the installation of the `eqn` and `troff` commands, which are unsupported.

–ek | –ke Preproceses the display with the `eqn` command, then performs the same steps as the `-k` option.

This option requires the installation of the `eqn`, `tc`, and `troff` commands, which are unsupported.

–k Formats the display through the `troff` command, using troff's `-t` option, then directs the output to the `tc` command.

This option requires the installation of the `tc` and `troff` commands, which are unsupported.

–n Formats the display through the `nroff` command. This is the default.

–ne | –en Preproceses the display with the `neqn` command, then performs the same steps as the `-n` option.

This option requires the installation of the `neqn` command, which is unsupported.

–t Photypesets the output through the `troff` command.

This option requires the installation of the `troff` command, which is unsupported.

–w Shows where the specified reference pages are located, relative to the `/usr/man` directory.

Restrictions

The reference pages are reproducible on phototypesetters or on hardcopy devices. However, some devices do not properly handle special characters which causes information to be lost.

Some options require the installation of unsupported software. Use of these options is at your own risk.

Options which call the neqn or eqn commands will generally fail when used with the ULTRIX reference pages, because any ULTRIX reference pages which use *eqn commands were preprocessed through the neqn text formatter before being packaged for shipment to you. *eqn text preprocessors generally report numerous errors when attempts are made to reprocess files a second time through an *eqn text preprocessor.

Both /usr/ucb/man and /usr/bin/man commands cd to the /usr/man directory before searching for and formatting files. Some reference pages assume that this happens. Therefore, an attempt to format some reference pages manually with a *roff text formatter may fail if you are not sitting in the /usr/man directory.

/usr/ucb/man Restrictions

If a specified reference page exists in the appropriate /usr/man/man? directory, but there is no appropriate /usr/man/cat? directory, you will not be able to scroll backwards in the display.

The man directories for sections n, l, o, p, 0 and 9 are optional directories. They must be created by the system administrator.

The /usr/man/cat? directories are not required to exist. They must be created by the system administrator. This is generally done through the catman(8) command.

Examples

/usr/ucb/man Examples

The following examples all assume the use of the default command: /usr/ucb/man.

The following example shows how to locate reference pages containing the keyword 'graph':

```
% man -k graph
```

The following example shows how to display the graph(1g) reference page:

```
% man 1g graph
```

The following example shows how to display plot reference pages:

```
% man 1 plot 3 plot
```

The following example shows how to display chmod and chown reference pages:

```
% man 1 chmod chown 2 chmod chown
```

The following example shows how to display a reference page test in the /usr/man/manl directory. In order to locate the test reference page here, it must have the file name test.l, so its reference page title would be test(l).

```
% man local test
```

To locate the test reference pages in Section 1:

```
% man 1 test 1sh5 test
```

If you have a directory /usr/local/man which contains man? subdirectories, which also contain reference pages, then the following example shows how to display a reference page games located somewhere in a subdirectory of /usr/local/man:

```
% man -P /usr/local games
```

/usr/bin/man Examples

The following example shows how to display chmod reference pages:

```
% /usr/bin/man chmod
```

The above displays all the chmod reference pages from all sections of the installed reference pages.

The following example shows how to display all the test reference pages in Section 1:

```
% /usr/bin/man 1 test
```

The following example shows how to locate all the test reference pages:

```
% /usr/bin/man -w test
```

The following example shows how to locate all the intro reference pages in Section 3:

```
% /usr/bin/man -w 3 intro
```

The following example displays the man(1) reference page with a starting page number of 10.

```
% /usr/bin/man -n10 1 man
```

Files

/usr/ucb/man	The default man command.
/usr/bin/man	The alternate man command.
/usr/man/man?/*	These directories contain the online reference pages which are divided into sections 1 through 8, n, l, o, and p. Sections 0 and 9 can also exist but these are non-standard sections.
/usr/man/cat?/*	These directories contain the files generated by the man and catman commands.
/usr/lib/whatis	This file contains the summary lines of each reference page.

manpath/man/man?/*
: These directories contain reference pages to be searched by the man command when the -P *manpath* option is specified. These directories must have the same organization and format as /usr/man.

See Also

apropos(1), col(1), man(7), nroff(1), page(1), tbl(1), whatis(1), whereis(1), catman(8)

Name

mark – mark messages

Syntax

mark [*+foldername*] [*msgs*] [**–sequence** *name...*] [**–add**] [**–delete**] [**–list**] [**–public**] [**–nopublic**] [**–zero**] [**–nozero**] [**–help**]

Description

Use the `mark` command to assign a name to a sequence of messages within the current folder. You can then use this message sequence with any `MH` command that takes a *msg* or *msgs* argument.

The following example shows how you can create a message sequence called "out" containing messages 10–20 in the current directory. The second part of the example shows how this sequence can be used in conjunction with the `rmm` command, to delete all the messages in the sequence.

```
% mark 10-20 -sequence out

% rmm out
```

You can specify a folder other than the current folder, by using the `<+folder>` argument.

Sequences still point to the same messages even if you sort all the messages in the folder with `sortm`.

If you delete a message or refile it in another folder, it is also deleted from the sequence.

You can use `mark` in conjunction with `pick` to give you a very powerful and flexible way to manipulate messages. The following example shows how you can combine the two commands together to select all messages from Adrian and put them in a sequence named Ateam. See `pick`(1mh) for more details of the power of `pick`.

```
mark `pick -from Adrian` -sequence Ateam
```

Note that you cannot use special characters, such as hyphens, in sequence names. Sequence names can consist of alphanumeric characters, but the first character must be alphabetic.

If you create a sequence using `mark`, the ordering of messages within the folder remains unchanged. So if messages 3, 7 and 9 are put into the sequence, they are still shown as messages 3, 7 and 9 if you use `scan` after incorporating them into the sequence. The `scan` command does not show any differences between messages that are in sequences and ordinary messages within a folder.

Options

If you use `mark` on its own without specifying `-sequence` *name*, it displays the sequences that have been created in the current folder. The following example shows how this works. It also illustrates that one message can be in more than one sequence at the same time.

```
% mark
cur: 19
one: 2 7 9
Two: 2-4
```

An identical result can be obtained if you use mark–list

A message sequence is a keyword, just like one of the reserved message names, such as first or next. Unlike the reserved message names, you can define, modify, and remove the semantics of a message sequence. Message sequences are folder-specific, for example: the sequence name seen in folder +inbox need not have any relation whatsoever to the sequence of the same name in a folder of a different name.

You can manipulate sequences with three options:

```
-add
-delete
-list
```

These switches are mutually exclusive: the last occurrence of any of them overrides any previous occurrence of the other two.

The -add switch tells mark to add messages to sequences or to create a new sequence. For each sequence named via the -sequence <name> argument the messages named in <msgs> (which defaults to the current message if no msgs are given), are added to the sequence. The messages to be added need not be absent from the sequence. If you specify the -zero option, all messages in the sequence are removed from the sequence, before the new messages are added to it. Note the messages are removed from the sequence only. They are not deleted or removed from your folder.

If you specify -add–nozero the specified messages are appended to the sequence.

The -delete switch tells mark to delete messages from sequences, and is the opposite of -add. For each of the specified sequences, the named messages are removed from the sequence. These messages need not be already present in the sequence. If the -zero switch is specified, then all the messages in the folder are appended to the sequence prior to the removal of the messages. The following example shows how this works in a folder with sixteen messages.

```
% mark -delete -zero 7 -sequence notseven

% mark
notseven: 1-6 8-16
```

Hence, -delete -zero means that each sequence should contain all messages except those indicated, while -delete -nozero means that only the indicated messages should be removed from each sequence. As expected, the command mark -sequence seen -delete all deletes the sequence seen from the current folder.

When creating (or modifying) a sequence, the -public switch indicates that the sequence should be made readable for other MH users. In contrast, the -nopublic switch indicates that the sequence should be exclusive to your MH environment.

The -list switch tells mark to list the sequences defined for the folder and the messages associated with those sequences.

You can list each sequence named by using the `-sequence name` switch. If you do not specify the sequence name, `-list` lists all sequences, and the messages associated with those sequences, in the specified folder. The `-zero` switch does not affect the operation of `-list`.

The name used to denote a message sequence must consist solely of alphabetic characters, and cannot be one of the reserved message names (such as, `first cur` and so forth).

You can define up to a maximum of 10 sequences in any one folder.

The name used to denote a message sequence cannot occur as part of a message range: for example, constructs like `seen:20` or `seen` –10 are forbidden.

The defaults for this command are:

`+folder` defaults to the current folder
`-add` if `msgs` is specified, `-list` otherwise
`msgs` defaults to cur (or all if `-list` is specified)
`-nopublic` if the folder is read–only, `-public` otherwise
`-nozero`

Files

$HOME/.mh_profile	Your user profile

Profile Components

Path:	To your MH directory
Current-Folder:	To find the default current folder

See Also

folder(1mh), pick(1mh), sortm(1mh)

Name

mdtar – multivolume archiver

Syntax

mdtar [*key*] [*name...*]

Description

The `mdtar` command saves multiple files on multiple archives (usually an RX50 diskette, but any file/device may be specified). `mdtar` actions are controlled by the **key** argument. The **key** is a string of characters containing one function letter and one or more function modifiers. Other arguments to `mdtar` are file or directory names specifying which files to dump or restore. In all cases, appearance of a directory name refers to the files and, recursively, subdirectories of that directory. `mdtar` also saves special files.

Options

–C Changes directory to specified `name`. This allows multiple directories not related by a close common parent, to be archived using short relative path names. For example, to archive files from `/usr/include` and from `/etc`, one might use

```
tar c -C /usr include . -C /etc .
```

The function portion of the key is specified by a letter.

c Creates a new archive. Writing begins at the beginning of the archive instead of after the last file.

r Writes the named files to the end of the archive.

t If no file argument is given, all Generates archive table of contents. If no argument is given, all of the names on the archive are listed. Produce a Table of contents.

u Updates the current archive. Adds the named files to the archive, if they are not there already or if they have been modified since last put on the archive.

x Extracts each specified file from the archive. If the named file matches a directory whose contents had been written onto the archive, this directory is recursively extracted. The owner, modification time, and mode are restored if you are the superuser and if you have also specified the **p** switch. If no file argument is given, the entire content of the archive is extracted. If multiple entries specifying the same file are on the archive, the last one overwrites previous versions.

The following characters may be used to qualify the function desired in addition to one or more of the above letters.

0...9 Selects unit number of the drive as an alternate disk drive. The default disk drive is the device named `/dev/rra1a`.

A Uses the specified number (next argument) as archive with which to

begin the output. This switch is intended for error recovery. mdtar outputs files in terms of Archives. Each Archive contains a number of files. If mdtar has been requested to dump a path (set of files) that consist of (for example) 10 archives and there is an error writing the nth Archive, then the A modifier may be used to restart mdtar at the nth Archive.

CAUTION

You must issue the same path (set of files) as in the first command. This will guarantee that mdtar will begin at the correct file on Archive n.

If the v mode is specified, mdtar outputs informational messages to inform the user of progress. For example, the following command will dump the entire directory structure:

```
mdtar cv
```

If an error occurs on Archive 7, to restart at the 7th Archive, without having to re-dump the first 6 Archives, issue the following command:

```
mdtar cvA 7
```

mdtar will tell you it is skipping the first 6 Archives and will resume output with the data that begins Archive 7.

b Uses the specified number (next argument) as the blocking factor. The default is 20 (the maximum is 20).

B Forces output blocking to 20 blocks per record.

f Uses the specified file (next argument) as the name of the archive. If the name of the file is –, tar writes to standard output (piping).

F[F] Operates in fast mode. When **F** is specified, mdtar skips all SCCS directories, core files, and errs files. When **FF** is specified, mdtar also skips all a.out and *.o files.

h Saves a copy of the file (excludes symbolic links). The default action of mdtar is to place symbolic link information on the output device. A copy of the file IS NOT saved on the output device.

i Ignores checksum errors found in the archive.

l Displays an error message if all links to the files dumped cannot be resolved. If -l is not specified, no error messages are printed.

m Does not restore file modification times. The modification time is the time of extraction. Normally, mdtar restores modification times of regular and special files.

o Suppresses the normal directory information. On output, mdtar normally places information specifying owner and modes of directories in the archive. Former versions of tar, when encountering this information will give the error message

```
<name>/: cannot create.
```

p Restores the named files to their original modes, ignoring the present umask(2). Setuid and sticky information will also be restored to the super-user. You must be Superuser to perform this option. For further

information, see stat(2), S_ISVTX.

s Uses specified number (next argument) as size of media in 512-byte blocks. This enables mdtar to be used with devices of different physical media sizes. The default is 800 blocks (assumption is an RX50 output Archive).

v Displays detailed (verbose) information as it archives files. Normally mdtar does its work silently. With the **t** function, the verbose option gives more information about the archive entries than just their names.

```
#cd
#tar cvf tar-out vmunix
```

Produces the output "a vmunix 1490 blocks" where 1490 is the number of 512 byte blocks in the file "vmunix".

```
#tar xvf tar-out
```

Produces the output "x vmunix, 762880 bytes, 1490 blocks" where 762880 is the number of bytes and 1490 is the number of 512 byte blocks in the file "vmunix" which was extracted.

w Displays action to be taken for each file and prompts for confirmation. If a word beginning with 'y' is given, the action is done. Any other input means do not do it.

Restrictions

The **u** option can be slow.

The current limit on file name length is 100 characters.

There is no way to follow symbolic links selectively.

Diagnostics

Indicates bad key characters and archive read/write errors.

Indicates if enough memory is not available to hold the link tables.

Files

/tmp/tar*

See Also

stat(2), tar(1)

Name

mesg – allow or disallow messages

Syntax

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

Description

The `mesg` command with argument **n** forbids messages via `write` and `talk` by revoking non-user write permission on the user's terminal. The `mesg` command with argument **y** reinstates permission. All by itself, `mesg` reports the current state without changing it.

Diagnostics

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

Files

/dev/tty*

See Also

talk(1), write(1)

mh(1mh)

Name

mh – Message Handler

Description

MH consists of a collection of fairly simple single-purpose programs to send, receive, save, and retrieve messages. See the individual MH reference pages for descriptions of the programs that make up MH.

Unlike `mail`, MH is an optionally installed basesystem component. You can freely intersperse MH commands with other shell commands that allow you to read and answer your mail while you are performing another task.

To get started using MH, edit either the `.profile`, `.login`, or `.cshrc` files in your home directory to add the pathname `/usr/new/mh` to your `$PATH`. Check the manual entry for the shell you use, if you do not know how to do this. Run the `inc` command, if you have never used MH before. The `inc` command creates the necessary default files and directories after prompting you. It also moves mail from your system maildrop into your MH `+inbox` folder. Each message is converted to MH format and stored as separate files in your `+inbox` folder until you read it. When you read a message, you can refile it in another file that you create.

Folders are directories in which messages are stored; the folders themselves are stored in your Mail directory. They are similar to folders in a normal office filing system. See `refile`(1mh) and `folder`(1mh) for more details. For each message it processes, `inc` prints one line only. The one-line display contains the `From:` field, the `Subject:` field and as much of the first line of the message as it can accommodate. The first message that `inc` processes becomes your current message. The current message is the message that all MH commands operate on unless you have specified the *msg* argument. You must run `inc` each time you want to incorporate new mail into your MH file.

The `scan` command prints a list of the messages in your current folder.

The commands `show`, `next` and `prev` are used to read specific messages from the current folder. Of these, `show` displays the current message. You can also display a specific message by specifying its number, which you pass as an argument to `show`. In the following example, the contents of message number 10 in the current folder will be displayed.

```
% show 10
```

The commands `next` and `prev` respectively display, the message numerically following or preceding the current message. In all cases, the message displayed becomes the current message. If there is no current message, `show` may be called with an argument, or `next` may be used to advance to the first message. The command `rmm` (remove message) deletes the current message.

You can delete messages other than the current message by specifying the message number or the message numbers. When you specify more than one message, you separate each message number by a space. In the following example, messages 2, 4 and 6 in the current folder are deleted.

```
% rmm 2 4 6
```

The command `repl` is used to respond to the current message (by default). It places you in the editor with a prototype response form. While you are in the editor, you may peruse the item you are responding to by reading the file `@`.

The `comp` command allows you to compose a message by putting you in the editor on a blank message header form, and then lets you send it.

All the `MH` commands may be run with the single argument, `-help`, which causes them to print a list of the arguments with which they may be invoked.

Commands which take a message number as an argument (`scan`, `show`, `repl`, also take one of the following words: `first`, `prev`, `cur`, `next`, or `last` to indicate (respectively) the first, previous, current, next, or last message in the current folder.

Commands, such as `rmm`, `scan`, or `show`, which take a range of message numbers also take any of the following abbreviations:

num1–num2 – Indicates all messages in the range *num1* to *num2*, inclusive. The specified range must contain at least one message.

num:+*n*
num:–*n* – Up to *n* messages beginning with (or ending with) message *num*. The value of *num* may be any of the `MH` message keywords: `first`, `prev`, `cur`, `next` or `last`.

`first:` *n*
`prev:` *n*
`next:` *n*
`last:` *n* - The first, previous, next or last *n* messages, if they exist.

There are many other possibilities, such as creating multiple folders for different topics, and automatically refiling messages according to subject, source, destination, or content. See the individual Reference Pages for more details of the rest of the `MH` commands.

Following is a list of all the `MH` commands:

ali(1mh)	–list mail aliases
anno(1mh)	–annotate messages
burst(1mh)	–explode digests into messages
comp(1mh)	–compose a message
dist(1mh)	–redistribute a message to additional addresses
folder(1mh)	–set/list current folder/message
folders(1mh)	–list all folders
forw(1mh)	–forward messages
inc(1mh)	–incorporate new mail
mark(1mh)	–mark messages
mhl(1mh)	–produce formatted listings of MH messages
mhmail(1mh)	–send or read mail
mhpath(1mh)	–print full pathnames of MH messages and folders
msgchk(1mh)	–check for messages
msh(1mh)	–MH shell
next(1mh)	–show the next message
packf(1mh)	–compress a folder into a single file
pick(1mh)	–select messages by content
prev(1mh)	–show the previous message

prompter(1mh)	–prompting editor front end
rcvstore(1mh)	–incorporate new mail asynchronously
refile(1mh)	–file messages in other folders
repl(1mh)	–reply to a message
rmf(1mh)	–remove folder
rmm(1mh)	–remove messages
scan(1mh)	–produce a one line per message scan listing
send(1mh)	–send a message
slocal(1mh)	–receive mail hooks
show(1mh)	–show (list) messages
sortm(1mh)	–sort messages
whatnow(1mh)	–prompting front–end for send
whom(1mh)	–report who will receive a message when it is sent
mh-alias(5mh)	–alias file for MH message system
mh_format(5mh)	–format file for MH message system
mh-mail(5mh)	–message format for MH message system
mh-profile(5mh)	–user customization for MH message system
mtstailor(5mh)	–system customization for MH
ap(8mh)	–parse addresses RFC 822–style
conflict(8mh)	–search for alias/password conflicts
dp(8mh)	–parse dates RFC 822–style
install-mh(8mh)	–initialize the MH environment
post(8mh)	–deliver a message

Files

/usr/new/mh	directory containing commands
/usr/new/lib/mh	MH library

Name

mhl – produce formatted listings of MH messages

Syntax

/usr/new/lib/mh/mhl [–bell] [–nobell] [–clear] [–noclear] [–folder *+foldername*] **[–form** *formfile*] **[–length** *lines*] **[–width** *columns*] **[–moreproc** *program*] **[–nomoreproc] [files ...] [–help]**

Description

The `mhl` command is a program for listing formatted messages and it can be used as a replacement for `more` (the default `showproc`).

As with `more`, each of the messages specified as arguments (or the standard input) are output. If more than one message file is specified, you are prompted prior to each one, and a <RETURN> or <EOT> begins the output, with <RETURN> clearing the screen (if appropriate), and <EOT> (usually CTRL–D) suppressing the screen clear. An <INTERRUPT> (usually CTRL–C) aborts the current message output, prompting for the next message, if there is one, and a <QUIT> (usually CTRL–E) terminates the program without generating a core dump.

Options

The `-bell` option tells `mhl` to ring the terminal bell at the end of each page, while the `-clear` option tells `mhl` to clear the screen at the end of each page, or output a formfeed after each message. Both of these switches, and their inverse counterparts, take effect only if the profile entry `moreproc` is defined but empty, and if `mhl` is outputting to a terminal. If the `moreproc` entry is defined and non-empty, and `mhl` is outputting to a terminal, then `mhl` causes the `moreproc` to be placed between the terminal and `mhl`, and the switches are ignored. Furthermore, if the `-clear` switch is used and `mhl's` output is directed to a terminal, then `mhl` consults the `$TERM` and `$TERMCAP` environment variables to determine your terminal type in order to find out how to clear the screen. If the `-clear` switch is used and `mhl's` output is not directed to a terminal (for example, a pipe or a file), then `mhl` sends a formfeed after each message.

To override the default `moreproc` and the profile entry, use the `-moreproc program` switch. Note that `mhl` never starts a `moreproc` if invoked on a hardcopy terminal.

The `-length length` and `-width width` switches set the screen length and width, respectively. These default to the values indicated by `$TERMINFO`, if appropriate; otherwise they default to 40 and 80, respectively.

The default format file used by `mhl` is called `mhl.format` (which is first searched for in your `MH` directory, and then sought in the `/usr/new/lib/mh` directory). This can be changed by using the `-form formatfile` switch.

Finally, the `-folder +folder` switch sets the `MH` folder name, which is used for the `-messagename` switch described below. The environment variable `$mhfolder` is consulted for the default value which `show`, `next`, and `prev` initialize appropriately.

The `mhl` command operates in two phases: read and parse the format file then process each message (file). During the first phase, an internal description of the format is produced as a structured list. In the second phase, this list is traversed for each message, outputting message information under the format constraints from the format file.

The "mhl.format" form file contains information controlling screen clearing, screen size, wrap-around control, transparent text, component ordering, and component formatting. Also, a list of components that should be ignored may be specified, and a couple of special components are defined to provide added functionality. Message output is in the order specified by the order in the format file.

Each line of mhl.format has one of the formats:

```
;comment
:cleartext
variable[variable...]
component:[variable,...]
```

A line beginning with a semi colon (;) is a comment, and is ignored. A line beginning with a colon (:) is clear text, and is output exactly as is. A line containing only a colon (:) produces a blank line in the output. A line beginning with "component:" defines the format for the specified component, and finally, remaining lines define the global environment.

For example, the line:

width=80,length=40,clearscreen,overflowtext="***",overflowoffset=5

defines the screen size to be 80 columns by 40 rows, specifies that the screen should be cleared prior to each page, that the overflow indentation is 5, and that overflow text should be flagged with "***".

If variables or arguments follow a component, they apply only to that component, otherwise, their affect is global. Since the whole format is parsed before any output processing, the last global switch setting for a variable applies to the whole message if that variable is used in a global context (bell, clearscreen, width, length). All of the current variables and their arguments are shown in the following table.

Variable	Type	Semantics
width	integer	screen width or component width
length	integer	screen length or component length
offset	integer	positions to indent "component: "
overflowtext	string	text to use at the beginning of an overflow line
overflowoffset	integer	positions to indent overflow lines
compwidth	integer	positions to indent component text after the first line is output
uppercase	flag	output text of this component in all upper case
nouppercase	flag	do not use uppercase
clearscreen	flag/G	clear the screen prior to each page
noclearscreen	flag/G	do not clear the screen
bell	flag/G	ring the bell at the end of each page
nobell	flag/G	disable bell

component	string/L	name to use instead of "component" for this component
nocomponent	flag	do not output "component: " for this component
center	flag	center component on line (works for one-line components only)
nocenter	flag	do not center
leftadjust	flag	strip off leading white-space on each line of text
noleftadjust	flag	do not leftadjust
compress	flag	change newlines in text to spaces
nocompress	flag	do not compress
formatfield	string	format string for this component
addrfield	flag	field contains addresses
datefield	flag	field contains dates

To specify the value of integer-valued and string-valued variables, follow their name with an equals-sign and the value. Integer-valued variables are given decimal values, while string-valued variables are given arbitrary text bracketed by double-quotes. If a value is suffixed by `/G` or `/L`, then its value is useful in a global-only or local-only context (respectively). A line of the form:

```
ignores=component,...
```

specifies a list of components which are never output.

The component "MessageName" (case-insensitive) outputs the actual message name (file name) preceded by the folder name if one is specified or found in the environment. The format is identical to that produced by the `-header` option to `show`.

The component "Extras" outputs all of the components of the message which were not matched by explicit components, or included in the ignore list. If this component is not specified, an ignore list is not needed since all non-specified components are ignored.

If `-nocomponent` is not specified, then the component name is output as it appears in the format file.

The default format is:

```
: -- using template mhl.format - -
overflowtext="***",overflowoffset=5
leftadjust,compwidth=9
ignores=msid,msgid,message-id,received
Date;formatfield="%<(nodate{text})%{text}%|%(pretty{text})%>"
To:
cc;
:
From:
Subject:
:
extras:nocomponent
:
body:nocomponent,overflowtext=,overflowoffset=0,noleftadjust
```

The variable `formatfield` specifies a format string (see `mh-format`(5mh)). The variables *addrfield* and *datefield* (which are mutually exclusive), control the interpretation of the escapes.

By default, mhl does not apply any formatting string to fields containing address or dates (see mh-mail(5mh) for a list of these fields). Note that this results in faster operation since mhl must parse both addresses and dates in order to apply a format string to them. If desired, mhl can be given a default format string for either address or date fields (but not both). To do this, on a global line specify either the variable *addrfield* or the variable *datefield*, along with the variable *formatfield*.

The defaults for mhl are:

```
-bell
-noclear
-length 40
-width 80
```

Files

/usr/new/lib/mh/mhl.format	The message template
or <mh-dir>/mhl.format	Rather than the standard template
$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine your MH directory
moreproc:	Program to use as interactive front-end

See Also

more(1), show(1mh), ap(8mh), dp(8mh)

Name

mhmail – send or read mail

Syntax

mhmail [*addrs ...*] **[–body** *text*] **[–cc** *addrs ..*] **[–from** *addr*] **[–subject** *subject*] **[–help]**

Description

`mhmail` is intended as a replacement for the standard mail programs, bellmail and ucbmail. See `binmail`(1) and `mail`(1) for more details of these mail programs. When invoked without arguments, it simply invokes `inc` to incorporate new messages from the user's maildrop. When one or more users is specified, a message is read from the standard input and spooled to a temporary file. `mhmail` then invokes `post` with the name of the temporary file as its argument to deliver the message to the specified user.

Options

The `-subject` *subject* switch can be used to specify the `Subject:` field of the message. The `-body text` switch can be used to specify the text of the message; if it is specified, then the standard input is not read. Normally, addresses appearing as arguments are put in the `To:` field. If the `-cc` switch is used, all addresses following it are placed in the `cc:` field.

By using `-from addr`, you can specify the `From:` header of the draft. `post` fills in the `Sender:` header correctly.

Normally, people will use `comp` and `send` to send messages.

Files

/usr/new/mh/inc	Program to incorporate a maildrop into a folder
/usr/new/lib/mh/post	Program to deliver a message
/tmp/mhmail/*	Temporary copy of message

Profile Components

None

See Also

inc(1mh), post(8mh)

mhpath (1mh)

Name

mhpath – print full pathnames of MH messages and folders

Syntax

mhpath [*+foldername*] [*msgs*] **[–help]**

Description

Use the mhpath command to display the full pathname of the specified folder. If you do not specify a folder, mhpath will display the pathname of the current folder.

If you specify a message with its message number, mhpath displays the pathname of the specified message. In the following example, mhpath displays message number three in the folder +inbox.

```
$ mhpath +inbox 3
/r/phyl/Mail/inbox
```

You can also specify a number of messages, or a range of messages. The following examples demonstrate mhpath displaying the path name for messages two and five and also two to five, in the current folder.

```
$ mhpath 2 5
/r/phyl/Mail/inbox/2
/r/phyl/Mail/inbox/5

$ mhpath 2-5
/r/phyl/Mail/inbox/2
/r/phyl/Mail/inbox/3
/r/phyl/Mail/inbox/4
/r/phyl/Mail/inbox/5
```

If the top of the range that you specify is greater than the last message in the folder, mhpath displays as much of the specified range as possible. Additionally mhpath can take a keyword or a sequence name. The following keywords are acceptable: new, first, last, next, cur and all. The keywords first and last display the pathnames for the first or last message in the specified folder. Both these keywords can be used in conjunction with a number to display the pathnames for the first or last n messages. The following example displays the pathnames for the first 2 messages in the current folder.

```
$ mhpath first:2
/r/phyl/Mail/test/3
/r/phyl/Mail/test/5
```

The keyword new displays the pathname for the message after the last message in the folder. You cannot use new as part of a message range.

The keywords prev and next display the pathname for either the last message or the next message relative to the current message of the specified folder. The keyword cur displays the pathname of the current message in the specified folder.

You can use more than one keyword in the same mhpath command line. See the following example.

```
$ mhpath +test last new
/r/phyl/Mail/test/6
/r/phyl/Mail/test/7
```

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder

See Also

folder(1mh)

mkdir(1)

Name

mkdir – make a new directory

Syntax

mkdir –p *dirname...*

Description

The `mkdir` command creates specified directories in mode 777. The directories are then modified by `umask`(2), according to how you have set up `umask`. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

The `mkdir` command requires write permission in the parent directory.

Options

–p Create intermediate directories as required. If this option is not specified, the full path prefix of *dirname* must already exist.

See Also

`rm`(1)

Name

mkfifo – make fifo special files

Syntax

mkfifo *filename ...*

Description

The `mkfifo` command creates the 'fifo special files' named by its operand list. The operands are taken sequentially, in the order specified, and each 'fifo special file' is either completed or, in the case of an error or signal, not created at all. Unless interrupted, `mkfifo` will attempt to create all files specified. Error messages are written to standard error.

Each 'fifo file' is created with a mode of 666, read and write privileges for the user, group and other. The mode is modified by clearing those bits set in the process's file mode creation mask. See `umask`(2) for more information.

See Also

`mknod`(2), `stat`(2), `umask`(2)

mkstr (1)

Name

mkstr – create an error message file

Syntax

mkstr [–] *messagefile prefix file...*

Description

The `mkstr` command is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

The `mkstr` command will process each of the specified *files,* placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. A typical usage of `mkstr` would be:

```
mkstr pistrings xx *.c
```

This command would cause all the error messages from the C source files in the current directory to be placed in the file *pistrings* and processed copies of the source for these files to be placed in files whose names are prefixed with *xx.*

To process the error messages in the source to the message file `mkstr` keys on the string 'error("' in the input stream. Each time it occurs, the C string starting at the '"' is placed in the message file followed by a null character and a new-line character. The null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly `cat` the error message file to see its contents. The massaged copy of the input file then contains a `lseek` pointer into the file which can be used to retrieve the message, that is:

```
char   filename[] =  "/usr/lib/pi_strings";
int    file = -1;

error(a1, a2, a3, a4)
{
       char buf[256];

       Bif (efil < 0) {
              efil = open(efilname, 0);
              if (efil < 0) {
oops:
                     perror(efilname);
                     exit(2);
              }
       }
       if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
              goto oops;
       printf(buf, a2, a3, a4);
}
```

Options

– Places error messages at the end of specified message file.

See Also

xstr(1), lseek(2)

mktemp (1)

Name

mktemp – make a name for a temporary file

Syntax

mktemp [**–c**] [**–d** *directory_name*] [**–p** *prefix*]

Description

The `mktemp` command makes a name for the pathname of a temporary file and writes that name to standard output. The name will not duplicate that of an existing file. Subsequent calls to `mktemp` will only generate a new file name if all previously generated file names have been created by the user and still exist. Error messages are written to standard error.

The *directory_name* generated by `mktemp` is the concatenation of a directory name, a slash (/), a file prefix, a dot (.), a four digit number and a unique character.

The directory name is chosen as follows:

(1) If the **–d** option is specified, *directory_name* is used.

(2) Otherwise, if the TMPDIR environment variable is set and a string that would yield a unique name can be obtained using the value of that variable as a directory name, this value is used.

(3) Otherwise, `/tmp` is used.

The *prefix* is chosen as follows:

(1) If the **–p** option is specified, *prefix* is used.

(2) Otherwise, if the LOGNAME environment variable is set, it is used as the *prefix*.

(3) Otherwise, the user's login name is used.

Options

–c Causes `mktemp` to attemp to create a regular file using the generated (or created) name string. If file creation is successful, a zero length file is created with access permissions derived from the process's file mode creation mask, see `umask`(2). No attempt is made to create a file if the length of the generated (or created) name string exceeds 1023 characters. It is the user's responsibility to remove files created by use of this option.

–d *directory_name*
Causes *directory_name* to be used as the directory portion of the pathname. In this case, *directory_name* is used instead of TMPDIR and `/tmp`.

–p *prefix* Causes the string *prefix* to be used as the file's *prefix*. It is used instead of LOGNAM and the user's login name. If the *prefix* is longer the 249 characters, it will be silently truncated to that length before the concatenation of the suffix.

Environmental Variables

LOGNAME When the **–p** *prefix* option is not specified, the value of this variable is used as the *prefix* of the filename, if it exists.

TMPDIR When the **–d** *directory_name* option is not specified, the value of this variable is used instead of `/tmp`.

Restrictions

If the user does not have write permission in the directory specified, and error message is reported and `/tmp` is used in its place. The entire path name can not exceed 1023 characters, and the temporary file name can not exceed 255 characters. If the generated file name is too long it is truncated to fit before the suffix is added.

See Also

`stat`(2), `umask`(2), `mktemp`(3)

more(1)

Name

more, page – display file data at your terminal

Syntax

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

page *more options*

Description

The `more` filter allows you to examine a file one screenful of text at a time on a soft-copy terminal. It normally pauses after each screenful, printing --More-- at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user presses the space bar, another screenful is displayed.

Options

+linenumber Start up at *linenumber*.

+/pattern Start up two lines before the line containing the regular expression *pattern*. The command line options are:

–c Begins each page at the top of the screen and erases each line just before it draws on it. This avoids scrolling the screen, making it easier to read while `more` is writing. This option is ignored if the terminal does not have the ability to clear to the end of a line.

–d Displays extended continuation prompt at end of each display. The `more` command prompts the user with the message "Press space to continue, ´q´ to quit." at the end of each screenful, and responds to subsequent illegal user input by printing "Press ´h´ for instructions." instead of ringing the bell. This is useful if **more** is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

–f Counts logical text lines (does not fold long lines). This option is recommended if `nroff` output is being piped through `ul`, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen positions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus `more` may think that lines are longer than they actually are, and fold lines erroneously.

–l Ignores line feeds (CTRL/Ls) and normally, pauses at line feeds. If this option is not given, `more` pauses after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen is cleared before the file is printed.

–*n* Specifies number of line `more` displays.

–s Squeezes multiple blank lines from the output, producing only one blank line. Especially helpful when viewing `nroff` output, this option maximizes the useful information present on the screen.

–u Ignores all underlining in the data. If the terminal can perform

underlining or has a stand-out mode, `more` outputs appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The *–u* option suppresses this processing.

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.

The `more` command looks in the file /etc/termcap to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

The `more` command 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 places the command sequence which sets up the *MORE* environment variable in the .cshrc or .profile file.

If `more` is reading from a file, rather than a pipe, then a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when `more` pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

***i*<space>**	Display *i* more lines, (or another screenful if no argument is given)
^D	Display 11 more lines (a ''scroll''). If *i* is given, then the scroll size is set to *i*.
d	Same as ^D (control-D)
***i*z**	Same as typing a space except that *i*, if present, becomes the new window size.
***i*s**	Skip *i* lines and print a screenful of lines
***i*f**	Skip *i* screenfuls and print a screenful of lines
***i*b or *i*^B**	Skip back *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 or ?	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. of the last regular expression entered.
***i*n**	Search for the *i*-th occurrence
'	(single quote) Go to the point from which the last search started. If no

search has been performed in the current file, this command goes back to the beginning of the file.

!command Invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

***i*:n** skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense)

***i*:p** Skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then `more` goes back to the beginning of the file. If *i* doesn't make sense, `more` skips back to the first file. If `more` is not reading from a file, the bell is rung and nothing else happens.

:f Display the current file name and line number.

:q or :Q Exit from `more`

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

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\). The `more` command stops sending output, and displays the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type 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 data base
/usr/lib/more.help	Help file

See Also

csh(1), man(1), msgs(1), script(1), sh(1), environ(7)

Name

msgchk – check for messages

Syntax

msgchk [–nodate] [–notify *all/mail/nomail*] **[users ...] [–help]**

Description

The `msgchk` program checks all known mail drops for mail that is waiting for you. `msgchk` displays whether you have mail waiting to be read or not and shows the date that you last read your mail. You can suppress this display by specifying `-nodate` with `msgchk`. The following example shows `msgchk` in use.

```
$ msgchk
You have new mail waiting; last read on Tue, 07 Jun 88 17:21:49 WET
```

Options

The `-notify type` switch indicates under what circumstances `msgchk` should produce a message. The `type` switch can take values of `all, mail, nomail`. The default is `-notify all` which says that `msgchk` will report the status of the maildrop regardless of whether it has mail in it or not. The `mail` switch sets `msgchk` to report the status of the maildrop only if there is mail waiting. The `-nomail` switch sets `msgchk` to report the status of the maildrop only if there is no mail in it.

You can use `msgchk` to check on the status of other users' maildrops by specifying their user names. The following example illustrates this.

```
$ msgchk Rabb Jones
Rabb doesn't have any new mail waiting;
             last read on Tue 07 Jun 13:22:25 WET
Jones has new mail waiting; last read on Tue, 07 Jun 17:30:05 WET
```

Restrictions

`msgchk` does not understand the `$MAILDROP` envariable.

Files

$HOME/.mh_profile	The user profile
/usr/new/lib/mh/mtstailor	Tailor file
/usr/spool/mail/$USER	Location of mail drop

Profile Components

None

See Also

inc(1mh)

msh (1mh)

Name

msh – MH shell

Syntax

msh [–prompt *string*] **[–scan] [–noscan] [–topcur] [–notopcur] [file] [–help]**

Description

The command msh is an interactive program that implements a subset of the normal MH commands operating on a single file in packf format. That is, msh is used to read a file that contains a number of messages, as opposed to the standard MH style of reading a number of files, each file being a separate message in a folder.

The chief advantage of msh is that, unlike the normal MH style, it allows a file to have more than one message in it. In addition, msh can be used on other files, such as message archives which have been packed (see packf(1mh)).

When invoked, msh reads the named file, and enters a command loop. You can type most of the normal MH commands. The syntax and semantics of these commands typed to msh are identical to their MH counterparts. In cases where the nature of msh would be inconsistent with the way MH works (for example, specifying a *+folder* with some commands), msh will duly inform you. The commands that msh currently supports are:

```
ali      burst    comp    dist     folder
forw     inc      mark    mhmail   msgchk
next     packf    pick    prev     refile
repl     rmm      scan    send     show
sortm    whatnow  whom
```

In addition, msh has a help command which gives a brief overview of all the msh options. To terminate msh, either type CTRL-D, or use the quit command.

If the file is writable and has been modified, then using quit will ask you if the file should be updated.

Options

The -prompt *string* switch sets the prompting string for msh.

You may wish to use an alternative MH profile for the commands that msh executes; see mh-profile(5mh) for details of the H envariable.

A redirection facility is supported by msh. Commands may be followed by one of the following:

|	Open an interprocess channel – connect output to another ULTRIX command
>	Write output to file
>>	Append output to file

If *file* starts with a tilde (~), then a Cshell-like expansion takes place. Note that command is interpreted by sh(1). Also note that msh does **not** support history substitutions, variable substitutions, or alias substitutions.

When parsing commands to the left of any redirection symbol, msh will honor the backslash (\) as the quote next-character symbol, and double quote (") as quote-word delimiters. All other input tokens are separated by whitespace (spaces and tabs).

The following defaults are used by msh:

```
file defaults to ./msgbox
-prompt (msh)
-noscan
-notopcur
```

The argument to the -prompt switch must be interpreted as a single token by the shell that invokes msh. Therefore, you should place the argument to this switch inside double-quotes.

There is a strict limit of messages per file in packf format which msh can handle. Usually, this limit is 1000 messages.

Please remember that msh is not the cshell, and that a lot of the facilities provided by the latter are not present in the former.

In particular, msh does not understand back-quoting, so the only effective way to use pick inside msh is to always use the -seq select switch. If you put the line

```
pick: -seq select -list
```

in your mh_profile file, pick will work equally well from both the shell and msh.

Files

OME/.mh_profile	The user profile
/usr/new/lib/mh/mtstailor	tailor file

Profile Components

Path:	To determine your MH directory
Msg–Protect:	To set mode when creating a new CW file
fileproc:	Program to file messages
showproc:	Program to show messages

mt(1)

Name

mt – magnetic tape manipulating program

Syntax

mt [**–f** *tapename*] *command* [*count*]

Description

The `mt` command permits the operation of a magnetic tape drive.

Options

The **–f** flag option uses the specified tape device (next argument) in place of either that tape device defined by your TAPE environment variable (.login or .profile) or /dev/nrmt0h.

Some operations may be performed multiple times by specifying *count*. By default, `mt` performs the requested operation once.

The *command* argument defines the operation to be performed. Only as many characters as are required to uniquely identify a command need be specified.

The following is a list of commands:

bsf Backspace *count* files.

bsr Backspace *count* records.

cache Allows `mt` to use the cache buffer on a tape drive that has the cache buffer feature.

clhrdsf Clear hardware/software problem. Works with tape drives which use the TMSCP tape controller interface `tms`(4). This command is restricted to root access only.

clserex Clear serious exception. Works with tape drives which use the TMSCP tape controller interface `tms`(4).

clsub Clear subsystem. Works with tape drives which use the TMSCP tape controller interface `tms`(4). This command is restricted to root access only.

eof, weof Write *count* end-of-file marks at the current position on the tape.

eotdis Disable end-of-tape detection. When the end of tape is reached, the tape will run off the reel. Only the superuser can issue this command. The command remains in effect for the device until end-of-tape detection is enabled with the **eoten** command.

eoten Enable end-of-tape detection. When the end-of-tape markers are reached, the tape is halted on the reel, between the two end-of-tape markers. Only the superuser can issue this command. The command remains in effect for the device until end-of-tape detection is disabled with the **eotdis** command. This is the default mode after a system boot.

fsf	Forward-space *count* files.
fsr	Forward-space *count* records.
nocache	Disables the use of the cache buffer for any tape drive that has the cache buffer feature.
offline, rewoffl	Rewind the tape and place the tape unit off-line.
rewind	Rewind the tape.
status	Print status information about the tape unit.

Examples

This example shows how to rewind the tape `rmt0l`:

```
mt -f /dev/rmt0l rewind
```

This example shows how to backspace the tape nmt1h three files:

```
mt -f /dev/nrmt1h bsf 3
```

This example shows how to write two end-of-file marks at the current position on tape nmt6h:

```
mt -f /dev/nrmt6h eof 2
```

Return Value

In shell scripts, `mt` returns a 0 exit status when the operation(s) were successful, 1 if the command was unrecognized, and 2 if an operation failed.

Files

/dev/rmt?h or /dev/rmt?l
Raw magnetic tape interface with rewind when closed

/dev/nmt?h or /dev/nmt?l
Raw magnetic tape interface with no rewind when closed

See Also

dd(1), tar(1), ioctl(2), mtio(4), tms(4), environ(7)

mv(1)

Name

mv – move or rename files

Syntax

mv [**–i**] [**–f**] [**–**] *file1 file2*

mv [**–i**] [**–f**] [**–**] *file... directory*

Description

The mv command moves (changes the name of) *file1* to *file2*.

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, mv prints the mode and reads the standard input to obtain a line. If the line begins with **y**, the move takes place. If it does not, mv exits. For further information, see chmod(2).

In the second form, one or more *files* (plain files or directories) are moved to the *directory* with their original file-names.

The mv command refuses to move a file onto itself.

Options

–	Interprets all following arguments as file names to allow file names starting with a minus.
–f	Force. This option overrides any mode restrictions or the –i switch.
–i	Interactive mode. If a move is to supersede an existing file, the system prompts youw with the name of the file followed by a question mark. If you type a string that begins with y, the move occurs. If you type any other response, the move does not occur.

Restrictions

If *file1* and *file2* lie on different file systems, mv must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

See Also

cp(1), ln(1)

Name

nawk – data transformation, report generation language

Syntax

nawk [**–f** *programfile*] [**–F***s*] [*program*] [*var=value...*] [*file ...*]

Description

The nawk language is a file-processing language which is well-suited to data manipulation and retrieval of information from text files. This reference page provides a full technical description of nawk; if you are unfamiliar with the language, you will probably find it helpful to read the *Guide to the nawk Utility* before reading the following material.

A nawk program consists of any number of user-defined functions and 'rules' of the form:

pattern {*action*}

There are two ways to specify the nawk program:

(a) Directly on the command line. In this case, the *program* is a single command line argument, usually enclosed in apostrophes (').

(b) By using the **–f** *programfile* option (where *programfile* contains the nawk program). More than one **–f** option can appear on the command line. The program will consist of the concatenation of the contents of all the specified *programfiles*. You can use – in place of a file name, to obtain input from the standard input.

The input data manipulated by the nawk program is provided in *files* specified on the command line. If no such files are specified, data is read from the standard input. You can also specify a file name of – to mean the standard input.

Input to nawk is divided into *records*. By default, records are separated by new-line characters; however, you can specify a different record separator if you wish.

One at a time, and in order, each input record is compared with the pattern of every 'rule' in the nawk program. When a pattern matches, the action part of the rule is performed on the current input record. Patterns and actions often refer to separate *fields* within a record. By default, fields are separated by white space (blanks, new-lines, or horizontal tab characters); however, you can specify a different field separator string using the **–F***s* option (see **Input**).

You can omit the *pattern* or *action* part of a nawk rule (but not both). If *pattern* is omitted, the *action* is performed on every input record (as if every record matches). If *action* is omitted, every record matching the *pattern* will be written to the standard output.

If a line in a nawk program contains a '#' character, the '#' and everything after it is considered to be a comment.

Program lines can be continued by adding a backslash '\' to the end of the line. Statement lines ending with a comma ',', double or-bars '||', or double ampersands '&&', are automatically continued.

Options

-f *programfile*
Tells `nawk` to obtain its program from the specified file. There can be more than one of these on the command line.

-F*s* Says that *s* is the field separator character within records.

Variables and Expressions

There are three types of *variables* in `nawk`: *identifiers*, *fields*, and *array elements*.

An *identifier* is a sequence of letters, digits, and underscores beginning with a letter or an underscore.

Fields are described in the **Input** subsection.

Arrays are associative collections of values called the *elements* of the array. Array elements are referenced with constructs of the form

identifier[*subscript*]

where *subscript* has the form *expr* or *expr,expr,...* Each such *expr* can have any string value. Arrays with multiple *expr* subscripts are implemented by concatenating the string values of each *expr* with a separator character SUBSEP separating multiple *expr*. The initial value of SUBSEP is set to '\034' (ASCII field separator).

Fields and identifiers are sometimes called *scalar variables* to distinguish them from arrays.

Variables are not declared and need not be initialized. The value of an uninitialized variable is the empty string. Variables can be initialized on the command line using

var=value

Such initializations can be interspersed with the names of input files on the command line. Initializations and input files will be processed in the order they appear on the command line. For example, the command

```
nawk -f progfile A=1 f1 f2 A=2 f3
```

sets A to 1 before input is read from f1 and sets A to 2 before input is read from f3.

Certain built-in variables have special meaning to `nawk`, as described in later sections.

Expressions consist of constants, variables, functions, regular expressions and 'subscript in array' conditions (see below) combined with operators. Each variable and expression has a string value and a corresponding numeric value; the value appropriate to the context is used. If a string is used in a numeric context, and the contents of the string cannot be interpreted as a number, the 'value' of the string is taken to be zero.

Numeric constants are sequences of decimal digits.

String constants are quoted, as in **"x"**. Escape sequences accepted in literal strings are:

Escape	ASCII Character
\a	audible bell
\b	backspace
\f	formfeed

\n	new-line
\r	carriage return
\t	horizontal tab
\v	vertical tab
ooo	octal value *ooo*
\x*dd*	hexadecimal value *dd*
\"	quotation mark
c	any other character *c*

The regular expression syntax understood by nawk is the extended regular expressions of the egrep utility described in grep(1). Characters enclosed in slash characters '/' are compiled as regular expressions when the nawk program is read. In addition, literal strings and variables are interpreted as dynamic regular expressions on the right side of a '~' or '!~' operator, or as certain arguments to built-in matching and substitution functions. Note that when literal strings are used as regular expressions, extra backslashes are needed to escape regular expression metacharacters because the backslash is also the literal string escape character.

The 'subscript in array' condition is defined as:

index in *array*

where *index* looks like *expr* or *(expr,...,expr)*. This condition evaluates to 1 if the string value of *index* is a subscript of *array*, and to 0 otherwise. This is a way to determine if an array element exists. If the element does not exist, this condition will **not** create it.

Symbol Table

The symbol table can be accessed through the built-in array SYMTAB.

SYMTAB[*expr*]

is equivalent to the variable named by the evaluation of *expr*. For example,

SYMTAB["var"]

is a synonym for the variable *var*.

Environment

A nawk program can determine its initial environment by examining the ENVIRON array. If the environment consists of entries of the form:

name=value

then

ENVIRON[*name*]

has string value

"*value*"

For example, the following program is equivalent to the default output of env(1):

```
BEGIN   {
        for (i in ENVIRON)
                printf("%s=%s\n", i, ENVIRON[i])
        exit
}
```

Operators

The usual precedence order of arithmetic operations is followed unless overridden with parentheses; a table giving the order of operations appears at the end of the *Guide to the nawk Utility*. The unary operators are

-	Negation
+	Nothing (place holder)
--	Decrement by one
++	Increment by one

where the '++' and '--' operators can be used as either postfix or prefix operators, as in C.

The binary arithmetic operators are

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
^	Exponentiation

The conditional operator

expr ? *expr1* : *expr2*

evaluates to *expr1* if the value of *expr* is non-zero, and to *expr2* otherwise.

If two expressions are not separated by an operator, their string values are concatenated.

The operator '~' yields 1 (true) if the regular expression on the right side matches the string on the left side. The operator '!~' yields 1 when the right side has no match on the left. To illustrate:

```
$2 ~ /[0-9]/
```

selects any line where the second field contains at least one digit. Any string or variable on the right side of '~' or '!~' is interpreted as a dynamic regular expression.

The relational operators are the usual '<', '<=', '>', '>=', '==', and '!='.

The boolean operators are '||' (or), '&&' (and), and '!' (not).

Values can be assigned to a variable with

var = *expr*

If *op* is a binary arithmetic operator,

```
var op= expr
```

is equivalent to

```
var = var op expr
```

Command Line Arguments

The built-in variable ARGC is set to the number of command line arguments. The built-in array ARGV has elements subscripted with digits from zero to ARGC-1, giving command line arguments in the order they appeared on the command line.

The ARGC count and the ARGV vector do not include command line options (beginning with '–') or the program file (following `-f`). They do include the name of the command itself, initialization statements of the form

var=value

and the names of input data files.

The `nawk` language actually creates ARGC and ARGV before doing anything else. It then walks through ARGV processing the arguments. If an element of ARGV is the empty string, it is simply skipped. If it contains an equals sign '=', it is interpreted as a variable assignment. If it is a minus sign '–', it stands for the standard input and input is immediately read from the standard input until end-of-file is encountered. Otherwise, the argument is taken to be a file name; input will be read from that file until end-of-file is reached. Note that the program is executed by 'walking through' ARGV in this way; thus if the program changes ARGV, different files can be read and assignments made.

Input

Input is divided into records. Each record is separated from the next with a *record separator character*. The value of the built-in variable RS gives the current record separator character; by default, it begins as the new-line '\n'. If you assign a different character to RS, `nawk` will use that as the record separator character from that point on.

Records are divided into fields. Each field is separated from the next with a *field separator string*, given by the value of the built-in variable FS. You can set a specific separator string by assigning a value to FS or by specifying the **–F***s* option on the command line. FS can be be assigned a regular expression. For example,

```
FS = "[,:$]"
```

says that fields can be separated by commas, colons, or dollar signs. As a special case, assigning FS a string containing only a blank character sets the field separator to white space. In this case, any sequence of contiguous space and/or tab characters is considered a single field separator. This is the default for FS. However, if FS is assigned a string containing any other character, that character designates the start of a new field. For example, if we set

```
FS="\t"
```

(the tab character),

```
texta \t textb \t  \t  \t textc
```

contains five fields, two of which only contain blanks. With the default setting, the above would only contain three fields because the sequence of multiple blanks and tabs would be considered a single separator.

Various pieces of information about input are provided by the built-in variables listed below.

`NF`	Number of fields in the current record
`NR`	Number of records read so far
`FILENAME`	Name of file containing current record
`FNR`	Number of records read from current file

Field specifiers have the form $*i* where *i* runs from 1 through NF. Such a field specifier refers to the *i*th field of the current input record. **$0** (zero) refers to the entire current input record.

The **getline** function can read a value for a variable or **$0** from the current input, from a file, or from a pipe. The result of **getline** is an integer indicating whether the read operation was successful. A value of 1 indicates success; 0 indicates end-of-file encountered; and –1 indicates that an error occurred. Possible forms for **getline** are:

getline Reads next input record into **$0** and splits the record into fields. NF, NR, and FNR are set appropriately.

getline *var*
Reads next input record into the variable *var*. The record is not split into fields (which means that the current $*i* values do not change). NR and FNR are set appropriately.

getline <*expr*
Interprets the string value of *expr* to be a file name. The next record from that file is read into **$0** and split into fields. NF is set appropriately.

getline *var* <*expr*
Interprets the string value of *expr* to be a file name, and reads the next record from that file into the variable *var*. The record is not split into fields.

expr | **getline**
Interprets the string value of *expr* as a command line to be executed. Output from this command is piped into **getline**, and read into **$0** in a manner similar to **getline** <*expr*. See the SYSTEM FUNCTION section for additional details.

expr | **getline** *var*
Executes the string value of *expr* as a command and pipes the output of the command into **getline**. The result is similar to **getline** *var* <*expr*.

close(*expr*)
Only a limited number of files and pipes can be open at one time. This function will close open files or pipes. The *expr* must be one that came before '|' or after '>' in **getline**, or after '>', '>>', or '|' in **print** or **printf** as described in the **Output** section. By closing files and pipes that are no longer needed, you can use any number of files and pipes in the course of executing a `nawk` program.

Built-In Arithmetic Functions

int(*expr*)
Returns the integer part of the numeric value of *expr*. If (*expr*) is omitted, the integer part of **$0** is returned.

exp(*expr*), **log**(*expr*), **sqrt**(*expr*)
Returns the exponential, natural logarithm, and square root of the numeric value of *expr*. If (*expr*) is omitted, **$0** is used.

sin(*expr*), **cos**(*expr*)
Returns the sine and cosine of the numeric value of *expr* (interpreted as an angle in radians).

atan2(*expr1*, *expr2*)

Returns the arctangent of *expr1*/*expr2* in the range of $-\pi$ through π.

rand()
Returns a random floating-point number in the range 0 through 1.

srand(*expr***)**
Sets the seed of the **rand** function to the integer value of *expr*. If (*expr*) is omitted, `nawk` sets a default seed (which is the same each time `nawk` is invoked).

Built-In String Functions

len = **length(***expr***)**
Returns the number of characters in the string value of *expr*. If (*expr*) is omitted, **$0** is used.

n = **split(***string, array, regexp***)**
Splits the *string* into fields. The expression *regexp* is a regular expression giving the field separator string for the purposes of this operation. The elements of *array* are assigned the separated fields in order; subscripts for *array* begin at 1. All other elements of *array* are discarded. The result of **split** is the number of fields into which *string* was divided (which is also the maximum subscript for *array*). Note that *regexp* divides the record in the same way that the FS field separator string does. If *regexp* is omitted in the call to **split**, the current value of FS will be used.

str = **substr(***string, m, len***)**
Returns the substring of *string* that begins in position *m* and is at most *len* characters long. The first character of the string has *m* equal to one. If *len* is omitted, the rest of *string* is returned.

pos = **index(***s1, s2***)**
Returns the position of the first occurrence of string *s2* in string *s1*; if *s2* is not found in *s1*, **index** returns zero.

pos = *match*(*string, regexp*)
Searches *string* for the first substring matching the regular expression *regexp*, and returns an integer giving the position of this substring. If no such substring is found, **match** returns zero. The built-in variable RSTART is set to *pos* and the built-in variable RLENGTH is set to the length of the matched string. These are both set to zero if there is no match. The *regexp* can be enclosed in slashes or given as a string.

n = **gsub(***regexp, repl, string***)**
globally replaces all substrings of *string* that match the regular expression *regexp*, and replaces the substring with the string *repl*. If *string* is omitted, the current record (`$0`) is used. The notation **gsub** returns the number of substrings that were replaced or zero if no match occurred.

n = **sub(***regexp, repl, string***)**
Works like **gsub** except that at most one match and substitution is attempted.

str = **sprintf(***fmt, expr, expr...***)**
Formats the expression list *expr, expr, ...* using specifications from the string *fmt*, then returns the formatted string. The *fmt* string consists of conversion specifications which convert and add the next *expr* to the

string, and ordinary characters which are simply added to the string. Conversion specifications have the form

```
%[-][x][.y]c
```

where

- – left justifies the field
- *x* is the minimum field width
- *y* is the precision
- *c* is the conversion character

In a string, the precision is the maximum number of characters to be printed from the string; in a number, the precision is the number of digits to be printed to the right of the decimal point in a floating point value. If *x* or *y* is '*' (asterisk), the minimum field width or precision will be the value of the next *expr* in the call to **sprintf**.

The conversion character *c* is one of following:

- d Decimal integer
- o Unsigned octal integer
- x Unsigned hexadecimal integer
- u Unsigned decimal integer
- f Floating point
- e Floating point (scientific notation)
- g The shorter of e and f (suppresses non-significant zeros)
- c Single character of an integer value
- s String

n = **ord***(expr)*

Returns the integer value of first character in the string value of *expr*. This is useful in conjunction with '%c' in **sprintf**.

str = **tolower***(expr)*

Converts all letters in the string value of *expr* into lower case, and returns the result. If *expr* is omitted, **$0** is used.

str = **toupper***(expr)*

Converts all letters in the string value of *expr* into upper case, and returns the result. If *expr* is omitted, **$0** is used.

The System Function

status = **system***(expr)*

Executes the string value of *expr* as a command. For example,

```
system("tail " $1)
```

calls the `tail`(1) command, using the string value of **$1** as the file that `tail` should examine. See the **Restrictions** section for a discussion of the execution of the command.

User-Defined Functions

You can define your own functions using the form

```
function name (parameter-list) {
        statements
}
```

A function definition can appear in the place of a *pattern* {*action*} rule. The *parameter-list* contains any number of normal (scalar) and array variables separated by commas. When a function is called, scalar arguments are passed by value, and array arguments are passed by reference. The names specified in the *parameter-list* are local to the function; all other names used in the function are are global. Local scalar variables can be defined by adding them to the end of the parameter list. These extra parameters are not used in any call to the function.

A function returns to its caller either when the final statement in the function is executed, or when an explicit **return** statement is executed.

Patterns and Actions

A *pattern* is a regular expression, a special pattern, a pattern range, or any arithmetic expression.

BEGIN is a special pattern used to label actions that should be performed before any input records have been read. **END** is a special pattern used to label actions that should be performed after all input records have been read.

A pattern range is given as

pattern1, *pattern2*

This matches all lines from one that matches *pattern1* to one that matches *pattern2*, inclusive.

If a pattern is omitted, or if the numeric value of the pattern is non-zero (true), the resulting action is executed for the line.

An *action* is a series of statements terminated by semicolons, new-lines, or closing braces. A *condition* is any expression; a non-zero value is considered true, and a zero value is considered false. A *statement* is one of the following:

expression

```
if (condition)
        statement
[else
        statement]

while (condition)
        statement

do
        statement
while (condition)

for (expression1; condition; expression2)
        statement
```

The **for** statement is equivalent to:

```
expression1
while (condition) {
```

```
        statement
        expression2
}
```

The **for** statement can also have the form

```
for (i in array)
        statement
```

The *statement* is executed once for each element in *array*; on each repetition, the variable *i* will contain the name of a subscript of *array*, running through all the subscripts in an **arbitrary** order. If *array* is multi-dimensional (has multiple subscripts), *i* will be expressed as a single string with the SUBSEP character separating the subscripts. The following simple statements are supported:

`break` Exits a **for** or a **while** loop immediately.

`continue`
: Stops the current iteration of a **for** or **while** loop and begins the next iteration (if there is one).

`next` Terminates any processing for the current input record and immediately starts processing the next input record. Processing for the next record will begin with the first appropriate rule.

`exit[ (`*expr*`) ]`
: Immediately goes to the **END** action if it exists; if there is no **END** action, or if `nawk` is already executing the **END** action, the `nawk` program terminates. The exit status of the program is set to the numeric value of *expr*. If (*expr*) is omitted, the exit status is 0.

`return [`*expr*`]`
: Returns from the execution of a function. If an *expr* is specified, the value of the expression is returned as the result of the function. Otherwise, the function result is undefined.

`delete` *array*`[`*i*`]`
: Deletes element *i* from the given *array*.

`print` *expr*, *expr*, ...
: Described below.

`printf` *fmt*, *expr*, *expr*, ...
: Described below.

Output

The **print** and **printf** statements write to the standard output. Output can be redirected to a file or pipe as described below.

If >*expr* is added to a **print** or **printf** statement, the string value of *expr* is taken to be a file name, and output is written to that file. Similarly, if >RI >> expr is added, output will be appended to the current contents of the file. The distinction between '>' and '>>' is only important for the first **print** to the file *expr*. Subsequent outputs to an already open file will append to what is there already.

In order to eliminate ambiguities, statements such as

```
print a > b c
```

are syntactically illegal. Parentheses must be used to resolve the ambiguity.

If |*expr* is added to a **print** or **printf** statement, the string value of *expr* is taken to be an executable command. The command is executed with the output from the statement piped as input into the command.

As noted earlier, only a limited number of files and pipes can be open at any time. To avoid going over the limit, you should use the **close** function to close files and pipes when they are no longer needed.

The **print** statement prints its arguments with only simple formatting. If it has no arguments, the current input record is printed in its entirety. The output record separator ORS is added to the end of the output produced by each **print** statement; when arguments in the **print** statement are separated by commas, the corresponding output values will be separated by the output field separator OFS. ORS and OFS are built-in variables whose values can be changed by assigning them strings. The default output record separator is a new-line and the default output field separator is a space. The format of numbers output by **print** is given by the string OFMT. By default, the value is '%.6g'; this can be changed by assigning OFMT a different string value.

The **printf** statement formats its arguments using the *fmt* argument. Formatting is the same as for the built-in function **sprintf**. Unlike **print**, **printf** does not add output separators automatically. This gives the program more precise control of the output.

Restrictions

The longest input record is restricted to 20,000 bytes and the maximum number of fields supported is 4000. The length of the string produced by **sprintf** is limited to 1024 bytes.

The **ord** function may not be recognized by other versions of `awk`. The **toupper** and **tolower** functions and the ENVIRON array variable are found in the Bell Labs version of `awk`; this version is a superset of 'New `AWK`' as described in *The AWK Programming Language* by Aho, Weinberger, and Kernighan.

The *shell* that is used by the functions

```
getline    print    printf    system
```

and the return value of the **system** function is described in `system`(3).

Examples

The following example outputs the contents of the file `input1` with line numbers prepended to each line:

```
nawk '{print NR ":" $0}' input1
```

The following is an example using *var=value* on the command line:

```
nawk '{print NR SEP $0}' SEP=":" input1
```

The `nawk` program script can also be read from a file as in the command line:

```
nawk -f addline.nawk input1
```

This example produces the same output as the previous example when the file `addline.nawk` contains

```
{print NR ":" $0}
```

The following program appends all input lines starting with 'January' to the file `jan` (which can already exist or not), and all lines starting with 'February' or 'March' to the file `febmar`:

```
/^January/ {print >> "jan"}
/^February|^March/ {print >> "febmar"}
```

This program prints the total and average for the last column of each input line:

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

The following program interchanges the first and second fields of input lines:

```
{
        tmp = $1
        $1 = $2
        $2 = tmp
        print
}
```

The following example inserts line numbers so that output lines are left-aligned:

```
{printf "%-6d: %s\n", NR, $0}
```

This example prints input records in reverse order (assuming sufficient memory):

```
{
        a[NR] = $0 # index using record number
}
END {
        for (i = NR; i>0; --i)
                print a[i]
}
```

The next program determines the number of lines starting with the same first field:

```
{
        ++a[$1] # array indexed using the first field
}
END {   # note output will be in undefined order
        for (i in a)
                print a[i], "lines start with", i
}
```

The following program can be used to determine the number of lines in each input file:

```
{
        ++a[FILENAME]
}
END {
        for (file in a)
                if (a[file] == 1)
                        print file, "has 1 line"
                else
                        print file, "has", a[file], "lines"
}
```

This program illustrates how a two dimensional array can be used in nawk. Assume the first field contains a product number, the second field contains a month number, and the third field contains a quantity (bought, sold, or whatever). The program generates a table of products versus month.

```
BEGIN   {NUMPROD = 5}
{
        array[$1,$2] += $3
}
END     {
        print "\t Jan\t Feb\tMarch\tApril\t May\t" \
            "June\tJuly\t Aug\tSept\t Oct\t Nov\t Dec"
        for (prod = 1; prod <= NUMPROD; prod++) {
                printf "%-7s", "prod#" prod
                for (month = 1; month <= 12; month++){
                        printf "\t%5d", array[prod,month]
                }
                printf "\n"
        }
}
```

As this program reads in each line of input, it reports whether the line matches a pre-determined value:

```
function randint() {
        return (int((rand()+1)*10))
}
BEGIN   {
        prize[randint(),randint()] = "$100";
        prize[randint(),randint()] = "$10";
        prize[1,1] = "the booby prize"
        }
{
        if (($1,$2) in prize)
                printf "You have won %s!\n", prize[$1,$2]
}
END
```

This example prints lines whose first and last fields are the same, reversing the order of the fields:

```
$1==$NF {
        for (i = NF; i > 0; --i)
                printf "%s", $i (i>1 ? OFS : ORS)
}
```

The following program prints the input files from the command line. The **infiles** function first empties the array passed to it, and then fills the array. Notice that the extra parameter **i** of **infiles** is a local variable.

```
function infiles(f,   i) {
        for (i in f)
                delete f[i]
        for (i = 1; i < ARGC; i++)
                if (index(ARGV[i],"=") == 0)
                        f[i] = ARGV[i]
}
BEGIN   {
        infiles(a)
        for (i in a)
                print a[i]
```

```
        exit
}
```

This example is the standard recursive factorial function:

```
function fact(num) {
        if (num <= 1)
                return 1
        else
                return num * fact(num - 1)
}
{ print $0 " factorial is " fact($0) }
```

The last program illustrates the use of **getline** with a pipe. Here, **getline** sets the current record from the output of the wc(1) command. The program prints the number of words in each input file.

```
function words(file,   string) {
        string = "wc " fn
        string | getline
        close(string)
        return ($2)
}
BEGIN   {
        for (i=1; i<ARGC; i++) {
                fn = ARGV[i]
                printf "There are %d words in %s.",
                    words(fn), fn
        }
}
```

See Also

ed(1), grep(1), sed(1), ex(1), system(3), ascii(7),
"Awk – A Pattern Scanning and Processing Language" *ULTRIX Supplementary Documents, Vol. II: Programmer*

Name

netstat – show network status

Syntax

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 displays the contents of network-related data structures symbolically. Depending on the options for the information presented, there are a number of output formats. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. The third form, with an *interval* specified, continuously displays the information regarding packet traffic on the configured network interfaces. If no options are specified, `netstat` displays the state of all active sockets from those using any of the protocols listed in `/etc/protocols`.

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

If an *interval* is specified, `netstat` display the information regarding packet traffic on the configured network interfaces continuously, pausing *interval* seconds before refreshing the screen.

There are a number of display formats, depending on the information presented. The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and, optionally, the internal state of the protocol.

Address formats are either of the form host.port or network.port, if a socket's address specifies a network but no specific host address. When known, the host and network addresses are displayed symbolically according to the data bases `/etc/hosts` and `/etc/networks`, respectively. If a symbolic name for an address is unknown, or if the **–n** option is specified, the address is printed in the Internet dot format. Refer to `inet`(3n) for more information regarding this format. Unspecified, or wildcard, addresses and ports appear as an asterisk (*).

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network address (currently Internet specific) of the interface and the maximum transmission unit (mtu) are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route (for example, 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 may 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.

Options

–A	Displays the address of any associated protocol control blocks; used for debugging.
–a	Displays the information for all sockets. Normally sockets used by server processes are not shown.
–f *address_family*	Limits statistics or address control block reports to those of the specified *address family*. Recognized address families are *inet,* for AF_INET, and *unix,* for AF_UNIX.
–h	Displays the state of the IMP host table.
–I *interface*	Shows information only about this interface. Used with an *interval* displayed below.
–i	Displays status information for autoconfigured interfaces . Interfaces statically configured into a system, but not located at boot time are not shown.
–m	Displays information for the memory management routines The network manages a private share of memory.
–n	Displays network addresses as numbers. Normally `netstat` interprets addresses and attempts to display them symbolically.
–r	Displays the routing tables. When –s is also present, shows routing statistics instead.
–s	Displays per-protocol statistics.
–t	Displays time until interface watchdog routine starts up (used only in conjunction with **–i** option).

See Also

iostat(1), vmstat(1), hosts(5), networks(5), protocols(5), services(5), trpt(8c)

Name

newaliases – rebuild the data base for the mail aliases file

Syntax

newaliases

Description

The `newaliases` command rebuilds the random access data base for the mail aliases file `/usr/lib/aliases`. It must be run each time `/usr/lib/aliases` is changed in order for the change to take effect.

See Also

aliases(5), sendmail(8)

newinv (1)

Name

newinv – update distribution kit master inventory

Syntax

/usr/sys/dist/newinv *mi-file input-path*

Description

The `newinv` command interactively maintains the master inventory files used for producing distributions in `setld` format. The program updates the master inventory for a product when changes are made to the hierarchy of files which are to be packaged in the subsets which constitute the product.

The product hierarchy is scanned to produce a list of component path names relative to *input-path*. The list of path names is processed against the *mi-file* to produce a list of those files which have been removed from the product hierarchy and a list of those files which have been added.

The user is then given an opportunity to intervene and direct the inventory maintenance by editing these lists. The user is placed in the editor with each list available for editing. The editor used is the one specified by the *EDITOR* environment variable. If *EDITOR* is not set, `vi` is used. When editing the list of files which have been removed from the product, the user is expected to verify that the removals were intentional, and confirm the intent by removing the associated record from the file. When editing the list of files which have been added to the product, the user is expected to provide flags and subset information for each new file, transforming the elements of the list into master inventory records.

Both of these lists are merged with the records for the files which have not been changed to produce a new copy of the master inventory file.

Arguments

mi-file	The pathname of the master inventory file to be processed. If no master inventory file exists, you must create an empty one before using the `newinv` command.
input-path	The name of the product hierarchy to be scanned for files belonging in the inventory. All files and directories found below the *input-path* will be processed as belonging in the inventory.

Restrictions

The default text editor if not specified in *$EDITOR* is `/usr/ucb/vi`.

Files in the product hierarchy cannot be excluded from the master inventory. Files can be blocked from being kitted in the final distribution kit by setting the subset field of the master inventory record to *NOSHIP*.

Examples

To update the master inventory file *ULT400.mi* from the hierarchy beginning at */var/kits/input,* type:

```
newinv ULT400.mi /var/kits/input
```

Diagnostics

newinv: where is *mi-file?*
The *mi-file* specified on the command line cannot be found.

input-path: **bad directory.**
The *input-path* directory specified on the command line does not exist.

Files

`mi-file.bkp` Backup copy of master inventory

`mi-file.dead` The list of files missing from the product.

`mi-file.extra`
The list of files new to the product.

`mi-file.join` Intermediate join file.

`mi-file.tmp` List of all files in the product.

See Also

kits(1), vi(1), stl_mi(5), environ(7), setld(8)
Guide to Preparing Software for Distribution on ULTRIX Systems

next(1mh)

Name

next – show the next message

Syntax

next [+*foldername*] **[–header] [–noheader] [–showproc** *program*] **[–noshowproc] [switches for** `showproc`] **[–help]**

Description

The command `next` displays the next message in the current folder if you do not specify a folder with the +*folder* argument. The next message is the one after the current message in the specified folder. If you specify a folder with `next`, that folder will become the current folder. The message that is shown will become the current message.

Like `show`, it passes any switches onto the program `showproc`, which is called to list the message. This command is very similar to `show` *next*. See `show`(1mh) for more information.

The defaults for `next` are:
+*folder* defaults to the current folder
`-header`

Files

$HOME/.mh_profile The user profile

Profile Components

Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder
showproc:	Program to show the message

See Also

show(1mh), prev(1mh)

Name

nice, nohup – execute a command at a lower priority

Syntax

nice [*–number*] *command* [*arguments*]

nohup *command* [*arguments*]

Description

The `nice` command executes *command* with low scheduling priority (Bourne Shell only). If the *number* argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 20. The default *number* is 10.

The super-user may run commands with priority higher than normal by using a negative priority, for example, '—10'.

The `nohup` command executes *command* immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. The `nohup` command should be invoked from the shell with an ampersand (&) in order to prevent it from responding to interrupts by or stealing the input from the next person who logs in on the same terminal. The syntax of `nice` is also different.

Options

–number	Increments the priority by a specified number up to a limit of 20. The default is 10.

Restrictions

The `nice` and `nohup` commands are particular to `sh`(1). If you use `csh`(1), then commands executed with an ampersand (&) are automatically immune to hangup signals while in the background. There is a built-in command `nohup` which provides immunity from terminate, but it does not redirect output to nohup.out.

The `nice` command is built into `csh`(1) with a slightly different syntax than described here. The form "nice +10" nices to positive nice, and "nice –10" can be used by the superuser to give a process more of the processor.

Diagnostics

The `nice` command returns the exit status of the subject command.

Files

nohup.out	standard output and standard error file under nohup

See Also

csh(1), getpriority(2), renice(8)

nidl(1ncs)

Name

nidl – Network Interface Definition Language Compiler

Syntax

nidl *filename* [*options*]

Description

The `nidl` compiler is for the Network Interface Definition Language (NIDL).

The *filename* argument is the pathname of an interface definition file, written in the C syntax of NIDL.

The compiler generates a header file, a client stub file, a server stub file, and a client switch file, all in C source code. The compiler derives the names of these output files from *filename* by replacing the suffix (the rightmost period and all subsequent characters) with extensions for the client stub, server stub, and client switch.

Options

–confirm Display the options chosen but do not compile anything. In displaying information about **–idir**, the compiler constructs the list of all directories it would use to resolve relative pathnames of imported files, not just the ones explicitly supplied. (If the list is empty, the compiler uses only the current directory.) This option is useful for viewing the 'idir list' and for viewing the default values for other options.

–cpp *pathname* Run the specified program instead of the default C preprocessor. You can use the **–confirm** option to view the default pathname.

–def *def1* [*def2* ...] Pass the specified definitions to the C preprocessor. A definition can take either of two forms: *symbol* or *symbol=value*.

–exts *cstub–ext, sstub–ext, cswtch–ext*
Set the extensions that the compiler uses to name the stub and switch files it generates. The text strings *cstub–ext*, *sstub–ext*, and *cswtch–ext* must be separated by commas, with no spaces; they are used as extensions for the client stub, the server stub, and the client switch, respectively. You can use the **–confirm** option to view the defaults.

–f77c Generate client switch code that is compatible with the ULTRIX `f77` compiler. The NIDL compiler appends an underscore (`_`) character to the name of each client switch routine, so that the routines can be called from FORTRAN programs generated by the `f77` compiler.

–f77s Generate server stub code that is compatible with the ULTRIX `f77` compiler. The NIDL compiler appends an underscore (`_`) character to the name of each manager routine that the stub calls, so that the stub can call routines

generated by the `f77` compiler.

–idir *directory1* [*directory2 ...*]
Use the specified directories as paths from which to resolve relative pathnames of imported files. The compiler generates an ordered list of these directories. By default, it prepends to this list your current working directory and appends the system `idl` directory. You can suppress this default by supplying the **–no_def_idir** option.

–m
Support multiple versions and multiple managers within a single server. This option allows a server to export more than one version of an interface ('multiple versions') and to implement an interface for more than one type ('multiple managers').

The compiler appends the version number to the interface name when it generates identifiers in the stub and header files. For example, the interface specifier for version 3 of the **foobar** interface would be **foobar_v3$if_spec**.

The server for an interface compiled with **–m** must use `rpc_$register_mgr` to register its managers. The server supplies the name of a manager EPV to `rpc_$register_mgr`; the manager code defines this EPV. If the server supports objects of several types, it must use `rpc_$register_object` to register each object. These registrations enable the RPC runtime library at the server host to dispatch incoming requests to the correct manager.

If you do not specify either **–m** or its counterpart, **–s**, the compiler assumes **–s** and issues a warning. However, this default may be removed or changed in future NIDL compilers. Even if your server exports only one version of its interface and contains only one manager, use the **–m** option, so that it will be easy for you to incorporate multiple versions and multiple managers later.

–no_cpp
Do not run the C preprocessor on the input file. If you specify this option, the NIDL compiler does not interpret any C preprocessor statements (such as `#include` statements) in the interface definition.

–no_def_idir
Do not prepend the current working directory or append the system `idl` directory to the list of directories constructed from *-idir* arguments. If you specify **–no_def_idir** without **–idir**, the compiler resolves pathnames of imported files only relative to the current working directory.

–no_stubs
Do not generate any stub or switch files. The NIDL compiler generates only header files and insert files.

–no_warn
Suppress warning messages.

–out *directory*
Place the generated files in *directory*. The default is the current working directory.

–s	Allow a server to export only a single version of an interface and to implement an interface for only a single type. This option requests the behavior of NIDL compilers before Version 1.5, which added support for multiple versions and multiple interfaces. (See the **–m** option.) The server for an interface compiled with –s must use `rpc_$register` to register its interfaces. If you do not specify either **–s** or its counterpart, **–m**, the compiler assumes –s and issues a warning. However, this default may be removed or changed in future NIDL compilers. Even if your server exports only one version of its interface and contains only one manager, use the **–m** option, so that it will be easy for you to incorporate multiple versions and multiple managers later.
–space_opt	Reduce the size of generated stub code, possibly at the expense of slower data marshalling.
–version	Display the version number of the NIDL compiler but do not compile anything or generate any output files.

See Also

uuid_gen(3ncs)
DECrpc Programming Guide

Name

nl – line numbering filter

Syntax

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 from 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, you can elect not to number header and footer lines while numbering blank lines in the body.

The start of logical page sections is signaled by input lines containing nothing but the following delimiter characters:

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless otherwise specified, `nl` assumes that the text it is reading is in the body of a single logical page.

Options

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named.

–b *type*
: Specifies which logical page body lines are to be numbered. The following are recognized *types* and their meaning: **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*.

  The default *type* for logical page body is **t** (text lines numbered).

–h *type*
: Same as **–b** *type* except for header. Default *type* for logical page header is **n** (no lines numbered).

–f *type*
: Same as **–b** *type* except for footer. Default for logical page footer is **n** (no lines numbered).

–p
: Do not restart numbering at logical page delimiters.

–v *start#*
: The initial value used to number logical page lines. Default is 1.

–i *incr*	The increment value used to number logical page lines. Default is 1.
–s sep	The character used in separating the line number and the corresponding text line. Default *sep* is a tab.
–w *width*	The number of characters used for the line number. Default *width* is 6.
–n *format*	The line numbering format. Recognized values are the following: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes suppressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).
–l *num*	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**, or **–fa** option is set). Default is **1**.
–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, you must type two backslashes (//).

Examples

```
nl -v10 -i10 -d!+ file1
```

This command numbers file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

See Also

pr(1)

Name

nm – name list dump of RISC object files

Syntax

nm [-adefghnopruvxABTV] [file1 ... filen]

Description

The **nm** command prints listings formats for the symbol and external sections of the symbol table. A *file* can be an object or an archive. If you do not specify a file, this command assumes **a.out.**

Options

The **–A** and **–B** options specify AT&T System V style output or Berkeley (4.3 BSD) style output, respectively. The default is Berkeley (4.3 BSD).

NOTE

Some options can change the version-specific defaults. These options change the meaning of overloaded flags after -A or -B is specified.

A normal Berkeley system produces the *address* or *value* field followed by a *letter* showing what section the symbol or external is in and the *name* of the symbol or external.

These section letters describe the information that **nm** generates:

N nil storage class, compiler internal usage

T external text

t local text

D external initialized data

d local initialized data

B external zeroed data

b local zeroed data

A external absolute

a local absolute

U external undefined

G external small initialized data

g local small initialized data

S external small zeroed data

s local small zeroed data

R external read only

r local read only

C common

E small common

V external small undefined

The standard System V format and the **–a** specified Berkeley format provide an expanded listing with these columns:

Name the symbol or external name

Value the value field for the symbol or external, usually an address or interesting debugging information

Class the symbol type

Type the symbol's language declaration

Size unused

Index the symbol's index field

Section the symbol's storage class

NOTE

Every effort was made to map the field's functionality into System V nomenclature.

The **nm** command accepts these options:

-a prints debugging information, effectively turning Berkeley into System V format

-b prints the value field in octal

-d prints the value field in decimal (the System V default)

-e prints external and statics only

-f produces full output—**nm** still accepts this old option, but ignores it

-h does not print headers

-n for System V, sorts external symbols by name (default for Berkeley), and for Berkeley, sorts all symbols by value

-o for System V, prints the value field in octal, and for Berkeley prepends the filename to each symbol—good for grepping through **nm** of libraries

-p prints symbols as they are found in the file (the System V default)

-r reverses the sense of a value or name sort

-u prints only undefined symbols

-v sorts external symbols by value

-x prints value field in hexadecimal (Berkeley default)

-T truncates long names, inserting an asterisk (*) as the last printed character

-V prints version information on stderr

Name

nm – print program's name list

Syntax

nm [*options*] [*file...*]

Description

The nm command prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive is produced. If no *file* is given, the a.out file is used.

Each symbol name is preceded by its value, which is blank if undefined, and a letter. The letter defines the symbol type:

U Undefined

A Absolute

T Text segment symbol

D Data segment symbol

B Bss segment symbol

C Common symbol

f File name

– For dbx(1) symbol table entries (see **–a** option below).

If the symbol is local (non-external), the letter is in lower case. The output is sorted alphabetically.

Options

–a Displays all symbols including debug symbol table.

–e Prints only global (external) symbols.

–f Displays all symbols including debug symbol table.

–g Prints only global (external) symbols.

–n Sorts numerically rather than alphabetically.

–o Prepends file or archive element name to each output line.

–p Prints symbolic table order and does not sort.

–r Sorts in reverse order.

–u Displays only undefined symbols.

Diagnostics

bad format	Indicates that the file is neither an archive file nor a .o file.
cannot open	Indicates that the file could not be opened.
invalid argument	Indicates that an invalid option was specified.

no name list	Indicates that the file does not contain a symbol table.
ran out of memory	Indicates that the string table in either the archive or .o file is too big.

See Also

ar(1), ar(5), a.out(5), stab(5)

Name

nroff – format text

Syntax

nroff [*option...*] [*file...*]

Description

The `nroff` command formats text in the named *files* for typewriter-like devices. For further information, see the appropriate *roff reference pages. The full capabilities of `nroff` are described in the Nroff/Troff User's Manual.

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (–) is taken to be a file name corresponding to the standard input.

Options

The options, which may appear in any order so long as they appear before the files, are:

–o*list*	Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N–* means from *N* to the end.
–n*N*	Number first generated page *N*.
–s*N*	Stop every *N* pages. The `nroff` command halts prior to every *N* pages (default *N*=1) to allow paper loading or changing, and resumes upon receipt of a new line.
–m*name*	Prepend the macro file `/usr/lib/tmac/tmac.name` to the input *files*.
–r*aN*	Set register *a* (one-character) to *N*.
–i	Read standard input after the input files are exhausted.
–q	Invoke the simultaneous input-output mode of the **rd** request.
–T*name*	Prepare output for specified terminal. Known *names* are the *tabname* files in `/usr/lib/term`. The default name is tab37 for the Teletype Corporation Model 37 terminal.
–e	Produce equally-spaced words in adjusted lines, using full terminal resolution.
–h	Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

Files

/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for `nroff`

See Also

col(1), tbl(1), term(5), man(7), me(7), ms(7), term(7)
"Nroff/Troff User's Manual" and "A TROFF Tutorial" *ULTRIX Supplementary Documents* Vol. I: General User

Name

nslookup – interactively query servers running BIND, or BIND and Hesiod together

Syntax

/usr/ucb/nslookup [*host*][*server*]
/usr/ucb/nslookup [–][*server*]

Description

The `nslookup` command queries the BIND/Hesiod servers. This command has two modes: interactive and non-interactive. Interactive mode allows you to query the BIND/Hesiod server for information about various hosts and domains. Non-interactive mode allows you to obtain just the name and Internet address of a host or domain.

Use interactive mode if you have no arguments to provide. In this case, `nslookup` queries the default BIND/Hesiod server. To specify a BIND/Hesiod server to query, the first argument should be a dash (–) and the second argument should be the name of the server.

Use non-interactive mode when the name of the host you are looking up is the first argument. The optional second argument specifies a BIND/Hesiod server. If you do not supply a second argument, the current BIND/Hesiod server is queried.

To terminate the `nslookup` command from within interactive mode, press <CTRL/D>. To terminate only the current lookup activity, press <CTRL/C>.

Non-interactive Options

host [*server*]
: The *host* option is the name of the host for which you are looking up information. If you do not specify a *server*, the default server is queried. You can specify the *server* by either name or IP address.

– [*server*]
: The – option returns the name and IP address of the default server, or the *server* you specify. It then places you in interactive mode. If you do not specify a *server*, the default server is queried. You can specify the *server* by either name or IP address.

Interactive Options

The command line length must be less than 80 characters. Any unrecognized command is interpreted as a host name. The following are the standard options:

host [*server*]
: Looks up information for *host* using the current default server or using *server* if it is specified.

server *server*
: Changes the default server to the *server* specified. This option uses the current default server.

lserver *server*
: Changes the default server to the *server* specified. This option uses the initial

default server to look up information about the server specified.

root Changes the default server to the server for the root of the domain name space specified. Currently, the host nic.ddn.mil is used.

finger [*name*] [> [>] *file*]
Connects with the finger server on the current host. The current host is defined when a previous lookup for a host was successful and returned address information. See the **set querytype**=*value* command. The *name* field is optional; but if used, it specifies a user name. You can use the > and >> options to redirect output to the file specified.

ls [**–adhlmst**] *domain* [> [>] *file*]
Lists the information available for the *domain* specified. The default output contains host names and their Internet addresses.

–a Lists aliases of hosts in the *domain,* CNAME entries.

–d Lists all entries in the *domain.*

–h Lists CPU and operating system information for the *domain,* HINFO entries.

–l Same as –d.

–m Lists mail exchangers in the *domain,* MX entries.

–s Lists well known services in the *domain,* WKS entries.

–t Lists Hesiod text information, TXT entries.

If you redirect the output to a file, hash marks are printed for every 50 records received from the server.

view *file*
Sorts and lists the output of the `ls` command with the `more` command.

help or **?**
Print a brief summary of the `nslookup` commands and options.

set *keyword*[=*value*]
Changes the set options that affect the lookups, except for keywords `all` and `ALL` which display information. Valid keywords are:

all Prints the current values of the options you can set, as well as information about the current default server.

ALL Prints the current values of the options you can set, as well as information about the current default server. In addition, the **ALL** option prints the server state information.

[no]debug Turns on debugging mode. Verbose information is printed about the packet sent to the server and the resulting answer.

The default is **nodebug ,** which you can abbreviate to **[no]deb.**

[no]defname
Appends the default domain name to every lookup. The default is **nodefname,** which you can abbreviate to **[no]def.**

[no]recurse
Tells the BIND/Hesiod server to query other servers if it does not

have the information. The default is **recurse,** and the abbreviation is **[no]rec.**

[no]vc Uses a TCP connection when sending requests to the server. The default is **novc,** and the abbreviation is **[no]v.**

domain=*name*
Changes the default domain to the domain *name* specified. The default domain name is appended to all lookup requests if the **defname** option is set. The default value is set in the `/etc/resolv.conf` file, which you can abbreviate to **do.**

class=*value*
Changes the class of information returned from a query to one of the following values:

IN	Internet (default)
HS	Hesiod
ANY	any

The abbreviation for the **class** option is **cl.**

querytype=*value*
Changes the type of information returned from a query to *value.* The following is a list of the most common values:

A	host Internet address (default)
CNAME	canonical name for an alias
MX	mail exchanger
NS	name server
PTR	host Internet name
SOA	Start of authority
TXT	A Hesiod data query
WKS	A well known service

The abbreviation for the **querytype** option is **q.**

retry=*number*
Sets the number of retries to the *number* specified. If a reply to a request is not received within a certain amount of time (changed with **set timeout**), the request is resent. The retry value controls how many times a request is to be resent before giving up. The default retry number is `2`, and the abbreviation for the **retry** option is **ret.**

root=*host* Changes the name of the root server to the *host* name specified. This affects the **root** command. The default is `nic.ddn.mil`, and the abbreviation is **ro.**

timeout=*number*
Changes the time-out interval for waiting for a reply to the *number* specified (in seconds). The default is 10 seconds, and the abbreviation for the **timeout** option is **t.**

Tutorial

The domain name space is tree-structured and has six top-level domains:

- ARPA (for ARPAnet hosts)

  The ARPA domain is currently one of the top-level domains, but is being phased out.
- COM (for commercial establishments)
- EDU (for educational institutions)
- GOV (for government agencies)
- ORG (for not for profit organizations)
- MIL (for MILNET hosts)

If you are looking for a specific host, you need to know something about the host's organization in order to determine the top-level domain it belongs to. For instance, if you want to find the Internet address of a host at UCLA, do the following:

1. Connect with the root server, using the `root` command. The root server of the name space has knowledge of the top-level domains.
2. Connect with a server for the `ucla.edu` domain. The domain name for UCLA, which is a university, is `ucla.edu`. To connect with this server, you can type:

   ```
   # nslookup
   >  server ucla.edu
   ```

   The response is the names of the hosts that act as servers for the domain `ucla.edu`. Note that the root server does not have information about `ucla.edu`, but knows the names and addresses of hosts that do. All future queries are sent to the UCLA BIND server.
3. Request information about a particular host in the domain, for example, `purple`. To do this, type the host name. To request a list of hosts in the UCLA domain, use the `ls` command. The `ls` command requires a domain name (in this case, `ucla.edu`) as an argument.

Note that if you are connected with a BIND server that handles more than one domain, all lookups for host names must be fully specified with its domain. For instance, the domain `harvard.edu` is served by `seismo.css.gov`, which also services the `css.gov` and `cornell.edu` domains. A lookup request for the host `novel` in the `harvard.edu` domain must be specified as `novel.harvard.edu`. However, you can use the **set domain=***name* and **set defname** commands to automatically append a domain name to each request.

After a successful lookup of a host, use the `finger` command to see who is on the system or to get information about a specific person. To get other information about the host, use the **set querytype=***value* command, which allows you to change the type of information obtained and request another lookup. The `finger` command requires that the information requested information be of type A, a host Internet address.

Hesiod Tutorial

If you have set up Hesiod on your ULTRIX system and would like to look at this information, you must use the `set class=`*value* and `set querytype=`*value* commands, where value is HS and TXT respectively.

The following example presumes that the `networks` database is set up to be distributed with BIND/Hesiod. The answer received from the `nslookup` command is that 128.45 is the network number for the network named `ethernet` in the networks.dec.com domain.

```
# nslookup
Default Server:  localhost.dec.com
Address:  127.0.0.1

> set cl=hs
> set q=txt
> ethernet.networks
Server:  localhost.dec.com
Address:  127.0.0.1

ethernet.networks.dec.com  ethernet:128.45
>
```

Diagnostics

If the lookup request was not successful, the `nslookup` command displays one of the following error messages:

Time-out
The server did not respond to a request after a certain amount of time (changed with **set timeout=***value*) and a certain number of retries (changed with **set retry=***value*).

No information
Depending on the query type set with the **set querytype** command, no information about the host was available, although the host name is valid.

Non-existent domain
The host or domain name does not exist.

Connection refused
The connection to the BIND/Hesiod server was refused.

Network is unreachable
The connection to the BIND/Hesiod server cannot be made at the current time.

Server failure
The BIND/Hesiod server found an internal inconsistency in its database and could not return a valid answer.

Refused
The BIND/Hesiod server refused to service the request.

Format error
The name server found that the request packet was not in the proper format. Contact your DIGITAL Field Service representative.

Files

`/var/dss/namedb` BIND server data file directory

`/var/dss/namedb/named.boot`
BIND server boot file

`/var/dss/namedb/hosts.db`
BIND primary server hosts file

`/var/dss/namedb/hosts.rev`
BIND primary server reverse address hosts file

`/var/dss/namedb/named.local`
BIND server local host reverse address host file

`/var/dss/namedb/named.ca`
BIND server cache file

`/etc/resolv.conf`
BIND data file

See Also

finger(1), more(1), nsquery(1), resolver(3), resolver(5), named(8)
Guide to the BIND/Hesiod Service

Name

nsquery – name server query command

Syntax

/usr/ucb/nsquery [**lookup**] [*host*] [*server*]

Description

The `nsquery` command provides an interface to obtain host name and address information.

If you specify *host*, the `nsquery` command obtains information about the specified host. If no host is specified, the `nsquery` command obtains information about the local host system.

If you specify *server*, the `nsquery` command queries the BIND server that you specify. If you do not specify a server, the `nsquery` command queries the default BIND server.

Options

lookup Retrieves the host name, Internet Protocol (IP) address, and aliases of the specified host. If no host or server is specified, the `nsquery` command obtains information about the local system from the default BIND server.

If you do specify the **lookup** option, the `nsquery` command obtains the information about the BIND server and host specified (or their defaults). If the system from which you issue the `nsquery` command is a BIND server, and you do not specify the **lookup** option, information about only that server is retrieved.

Files

`/var/dss/namedb` Directory containing BIND server data file

`/var/dss/namedb/named.boot`
BIND server boot file

`/var/dss/namedb/hosts.db`
Host database file containing name to address mapping for BIND primary server

`/var/dss/namedb/hosts.rev`
Host database file containing address to name mapping for BIND primary server

`/var/dss/namedb/named.local`
Local host database file containing address to name mapping for BIND server

`/var/dss/namedb/named.ca`
BIND server cache file

`/etc/resolv.conf`
BIND data file

See Also

nslookup(1), resolver(3), resolver(5), named(8)
Guide to the BIND/Hesiod Service

Name

ntp – query a clock running the Network Time Protocol daemon, ntpd

Syntax

/usr/etc/ntp [**–v**][**–s**][**–f**] *host1* | *IPaddress1* ...

Description

The `ntp` command is used to determine the offset between the local clock and a remote clock. It can also be used to set the local host's time to a remote host's time. The `ntp` command sends an NTP packet to the NTP daemon, `ntpd`, running on each of the remote hosts specified on the command line. The remote hosts must be running `ntpd`. When the `ntpd` daemon on the remote host receives the NTP packet, it fills in the fields (as specified in RFC 1129), and sends the packet back. The `ntp` command then formats and prints the results on the standard output.

NOTE

You can specify hosts by either host name or Internet address. The hosts that you specify must either exist in the `/etc/hosts` file, or in the master `hosts` database, if the database is being served to your system by BIND/Hesiod or Yellow Pages.

The default output shows the roundtrip delay of the NTP packet in seconds, the estimated offset between the local time and remote time in seconds, and the date in `ctime` format. See the `ctime`(3) reference page for more information.

The `-s` and `-f` options can be used to reset the time of the local clock. Use `ntp` with these options to initialize the system time prior to running the `ntpd` daemon.

Options

–v Specifies verbose output. The output shows the full contents of the received NTP packets, plus the calculated offset and delay.

–s Sets local clock to remote time. This only happens if the offset between the local and remote time is less than 1000 seconds. The local clock is not reset if the remote host is unsynchronized.

If you specify more than one host name on the command line, `ntp` queries each host in order, waiting for each host to answer or timeout before querying the next host. The local clock is set to the time of the first remote host that responds.

–f Forces setting local clock regardless of offset. The `-f` option must be used with `-s` option. The local clock is not reset if the remote host is unsynchronized.

Restrictions

Using the `-s` and `-f` options require that you be logged on as superuser.

Examples

The following is the default output to an `ntp` query about a remote host with an internet address of 555.5.55.5:

```
# /usr/etc/ntp 555.5.55.5

555.5.55.5: delay:1.845207 offset:-0.358460  Mon Mar 20 08:05:44 1989
```

The following is the verbose output to an `ntp` query about the same remote host:

```
# /usr/etc/ntp -v 555.5.55.5

Packet from: [555.5.55.5]
Leap 0, version 1, mode Server, poll 6, precision -10 stratum 1 (WWVB)
Synch Distance is 0000.1999  0.099991
Synch Dispersion is 0000.0000  0.000000
Reference Timestamp is a7bea6c3.88b40000 Tue Mar  7 14:06:43 1989
Originate Timestamp is a7bea6d7.d7e6e652 Tue Mar  7 14:07:03 1989
Receive Timestamp is   a7bea6d7.cf1a0000 Tue Mar  7 14:07:03 1989
Transmit Timestamp is  a7bea6d8.0ccc0000 Tue Mar  7 14:07:04 1989
Input Timestamp is     a7bea6d8.1a77e5ea Tue Mar  7 14:07:04 1989
555.5.55.5: delay:0.019028 offset:-0.043890  Tue Mar  7 14:07:04 1989
```

The fields are interpreted as follows:

Packet from: [*internet address*]
The address of the remote host from which this NTP packet was received.

Leap *n*
The leap second indicator. Non-zero if there is to be a leap second inserted in the NTP timescale. The bits are set before 23:59 on the day of insertion and reset after 00:00 on the following day.

version *n*
The NTP protocol version.

mode *type*
The NTP mode can be Server, Client, Symmetric Passive, Symmetric Active, or Broadcast. See RFC 1129 for more information on NTP modes.

Poll *x*
The desired poll rate of the peer in seconds as a power of 2. For example, if poll is equal to 6, that means that the poll rate is one message exchanged every 2**6 seconds.

Precision *x*
The precision of the remote host's clock in seconds as a power of 2. For example, if precision is equal to –10, that means that the precision is 2**–10. The `ntpd` daemon sets this automatically.

Stratum *n* (*source*)
The stratum of the clock in the NTP hierarchy, along with the source of the clock. The *source* is either the name of a reference standard (such as WWVB or GOES), or the Internet address of the clock that this clock references.

Synch Distance is *nn.nn nn.nn*
The values reported are used internally by `ntpd`.

Synch Dispersion is *nn.nn nn.nn*
The values reported are used internally by `ntpd`.

The next five timestamps are given as NTP fixed-point values, in both hexadecimal and `ctime`. The timestamps are set either by this NTP process, or by the remote host you are querying. These timestamps are used by the local host to calculate delay and offset for this query.

Reference Timestamp is *hex-timestamp ctime_string*
This specifies the last time the remote host clock was adjusted. (remote time)

Originate Timestamp is *hex-timestamp ctime_string*
This specifies when the NTP request was transmitted by the local host to the remote host. (local time)

Receive Timestamp is *hex-timestamp ctime_string*
This specifies when the NTP request was received at the remote host. (remote time)

Transmit Timestamp is *hex-timestamp ctime_string*
This specifies when the NTP response was transmitted by the remote host. (remote time)

Input Timestamp is *hex-timestamp ctime_string*
This specifies when the NTP response was received by the local host. (local time)

hostname: **delay:***time* **offset:***time*
This field summarizes the results of the query, giving the host name or internet address of the responding clock specified in the command line, the round-trip delay in seconds, and the offset between the two clocks in seconds (assuming symmetric round-trip times).

Diagnostics

The following error messages can be returned by NTP:

`*Timeout*`

hostname `is not responding`
May indicate that the `ntpd` daemon is not running on the remote host.

`No such host:` *hostname*
The `ntpd` daemon cannot resolve the specified host name in the `/etc/hosts` file. Check that the host exists in the `/etc/hosts` file, or that it exists in the master `hosts` database, if the database is being served to your system by BIND/Hesiod or Yellow Pages.

See Also

ctime(3), ntp.conf(5), ntpd(8), ntpdc(8)
RFC 1129—Internet time synchronization: The Network Time Protocol
Guide to System and Network Setup
Introduction to Networking and Distributed System Services

od(1)

Name

od – create file octal dump

Syntax

od [*options*] [*file*] [*offset*] [*label*]

Description

The od command displays *file*, or its standard input, in one or more dump formats as selected by the first argument. If the first argument is missing, **–o** is the default. Dumping continues until end-of-file.

Options

–a Interprets bytes as characters and display them with their ACSII names. If the **p** character is given also, then bytes with even parity are underlined. The **P** character causes bytes with odd parity to be underlined. Otherwise the parity bit is ignored.

–b Displays bytes as unsigned octal.

–c Displays bytes as ASCII characters. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, formfeed=\f, newline=\n, return=\r, tab=\t; others appear as 3-digit octal numbers. Bytes with the parity bit set are displayed in octal.

–d Displays short words as unsigned decimal.

–f Displays long words as floating point.

–h Displays short words as unsigned hexadecimal.

–i Displays short words as signed decimal.

–l Displays long words as signed decimal.

–o Displays short words as unsigned octal.

–s[*n*] Looks for strings of ASCII characters of *n* minimum length. By default, the minimum length is 3 characters.

–v Displays all data and indicates lines identical to the last line shown with an * in column 1.

–w[*n*] Specifies the number of input bytes to be interpreted and displayed on each output line. If **w** is not specified, 16 bytes are read for each display line. If *n* is not specified, it defaults to 32.

–x Displays short words as hexadecimal.

An upper case format character implies the long or double precision form of the object.

The *offset* argument specifies the byte offset into the file where dumping is to commence. By default this argument is interpreted in octal. A different radix can be specified; If "." is appended to the argument, then *offset* is interpreted in decimal. If *offset* begins with "**x**" or "**0x**", it is interpreted in hexadecimal. If "**b**" ("**B**") is appended, the offset is interpreted as a block count, where a block is 512 (1024)

bytes. If the *file* argument is omitted, an *offset* argument must be preceded by ''+''.

The radix of the displayed address is the same as the radix of the *offset*, if specified; otherwise it is octal.

The *label* is interpreted as a pseudo-address for the first byte displayed. It is shown in ''()'' following the file offset. It is intended to be used with core images to indicate the real memory address. The syntax for *label* is identical to that for *offset*.

Restrictions

A file name argument can't start with ''+''. A hexadecimal offset can't be a block count. Only one file name argument can be given.

It is an historical botch to require specification of object, radix, and sign representation in a single character argument.

See Also

adb(1) - VAX only, dbx(1)

Name

odump – dumps selected parts of an object file

Syntax

odump [*options*] *file...*

Description

The odump command dumps selected parts of each object *file*. This command works for object files and archives of object files.

Options

The following options are available with the odump command:

–a Dumps the archive header for each member of the specified archive file.

–f Dumps each file header.

–g Dumps the global symbols from the symbol table of a RISC archive.

–o Dumps each optional header.

–h Dumps section headers.

–i Dumps the symbolic information header.

–s Dumps section contents.

–r Dumps relocation information.

–l Dumps line number information.

–t Dumps symbol table entries.

–z*name* Dumps line number entries for the specified function *name*.

–c Dumps the string table.

–L Interpret and print the contents of the *.lib* sections.

–F Dumps the file descriptor table.

–P Dumps the procedure descriptor table.

–R Dumps the relative file index table.

The odump command accepts these modifiers with the options:

–d *number* Dumps the section number or a range of sections starting at *number* and ending either at the last section number or the *number* you specify with **+d.**

+d *number* Dumps sections in the range beginning with the first section or beginning with the section you specify with **–d.**

–n *name* Dumps information only about the specified *name*. This modifier works with **–h, –s, –r, –l,** and **–t.**

–p Does not print headers

–t *index* Dumps only the indexed symbol table entry. You can also specify a range of symbol table entries by using the modifier **–t** with the **+t** option.

+t *index* Dumps the symbol table entries in the specified range. The range begins at the first symbol table entry or at the entry specified by **–t.** The range ends with the specified indexed entry.

–u Underlines the name of the file for emphasis.

–v Dumps information symbolically rather than numerically (for example, Static rather than **0X02**). You can use **–v** with all the options except **–s.**

–z *name,number*
Dumps the specified line number entry or a range of line numbers. The range starts at the *number* for the named function.

+z *number* Dumps line numbers for a specified range. The range starts at either the *name* or *number* specified by **–z** The range ends with the *number* specified by **+z.**

Also, an option and its modifier can be separated by using blanks. The name can be separated from the number that modifies **–z** by replacing the comma with a blank.

The `odump` command tries to format information in a helpful way, printing information in character, hexadecimal, octal, or decimal as appropriate.

See Also

a.out(5), ar(5)

pack(1)

Name

pack, pcat, unpack – compress and expand files

Syntax

pack [–] [**–f**] *name...*

pcat *name...*

unpack *name...*

Description

The `pack` command stores 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 forces packing of *name*. Using this option you can cause an entire directory to be packed even if some of the files cannot benefit from it. If `pack` is successful, *name* is removed. Packed files can be restored to their original form using `unpack` or `pcat`.

The `pack` command uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If a hyphen (–) is used as an argument, 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 a hyphen (–) in place of *name* causes 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 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 are about 90% of the original size.

The `pack` command returns a value that is the number of files that it failed to compress.

No packing occurs if one of the following is true:

- The file appears packed.
- The file name exceeds 12 characters.
- The file has links.
- The file is a directory.
- The file cannot be opened.
- No disk storage blocks can 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 not exceed 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

The `pcat` command does for packed files what `cat`(1) does for ordinary files, except that `pcat` can not 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
```

The `pcat` command 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.*

The `unpack` command 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.

The `unpack` command returns a value that is the number of files it was unable to unpack. Failure occurs for the same reasons that it occurs in `pcat`, as well as for the following:

a file with the unpacked name already exists;
if the unpacked file cannot be created.

This command is present only for compatibility. In general, the `compress`(1) command runs faster and gives better compression.

See Also

cat(1), compress(1)

packf(1mh)

Name

packf – compress a folder into a single file

Syntax

packf [*+folder*] [*msgs*] **[–file** *name***] [–help]**

Description

Each message in a folder is normally stored as a separate file. The `packf` command takes all messages from the current folder and copies them to a single specified file. Each message in the file is separated by four <CTRL-A's> and a newline.

You can specify a folder other than the current folder using by the *+folder* argument. If you do not want all the messages in a folder to be packed into one file, you can specify a number of messages or a range of messages with message numbers. The following example show messages one and three, and one through three in the folder +lrp being packed into a file.

```
$ packf +lrp 1 3
Create file "/machine/disk/username/msgbox"? y

$ packf +lrp 1-3
```

Options

If you specify an existing file using the `-file` *filename* switch then the specified messages will be appended to the end of that file, otherwise a new file will be created and the messages appended. If you do not specify a filename, `packf` will create one for you.

If a folder is given, it will become the current folder. The first message packed will become the current message.

The default settings for `packf` are:

+folder defaults to the current folder
msgs defaults to all
`-file`

Files

$HOME/.mh_profile The user profile

Profile Components

Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder
Msg-Protect:	To set mode when creating a new file

Name

pagesize – print system page size

Syntax

pagesize

Description

The `pagesize` command prints the size of a page of memory in bytes, as returned by `getpagesize`(2). This program is useful in constructing portable shell scripts.

See Also

getpagesize(2)

passwd(1)

Name

passwd – create or change password

Syntax

passwd [–aefs] [*name*]

Description

The `passwd` command lets you or the superuser change your password. When you enter the `passwd` command, the program prompts you for the old password and then for the new password. Next, the program asks you for the new password again, to verify that you have typed it correctly. Note that the passwords are not displayed on the screen.

Your new password must meet the length requirements specified by the superuser. To review these requirements, refer to the `/etc/svc.conf` file. (This is a read-only file.)

If your system is running with increased security, you may have to choose a password from a list of randomly generated passwords, or you may need authorization to change your password. At higher security levels, you may be prohibited from changing your password until its minimum lifetime has expired, as specified in the Authorization Database.

If you are running the BIND/Hesiod service, your password will be updated automatically on the server.

Options

–a Supply a list of randomly generated passwords. (See the Examples section.)

–e Use an extended protocol when communicating with a prompter program. As a result, the standard input and output (used by `login`) is sent to this program.

–f Change the finger information, not the password. The finger program provides information about current ULTRIX users, such as login and terminal name, idle time and office location.

–s Change the login shell of the password file, not the password entry.

Restrictions

If you use a hardcopy terminal, you must destroy all print outs of valid passwords.

Examples

The following example illustrates the **–a** option, which displays a list of randomly generated passwords and their suggested pronunciation with hyphens. The hyphens delineate the syllables of the passwords:

```
passwd -a abcd
Changing password for abcd

Here are some suggested passwords:

ryegd           ryeg-di
aswurku         a-swurk-u
ryedok          ryed-ok
teleccs         tel-ec-cos
wahislas        wa-hi-slas

Enter new password:
```

Diagnostics

Password must be at least 6 characters long, password unchanged
Your password does not meet the minimum length requirement specified in `/etc/svc.conf`.

Warning: Only the first 8 characters of the password are significant
Your password exceeds the maximum length requirement specified in `/etc/svc.conf`.

Permission denied
You do not have the privilege to change your password. The minimum lifetime has not expired.

Password is not different enough, unchanged
Your new password must be different from your old password.

Password must be different than logname, and not resemble previous password
Your new password must be different from your login name.

Verification failed, password unchanged
You misspelled the verification of your new password.

Files

`/etc/passwd`	Password file
`/etc/auth.dir`	Authorization data base directory
`/etc/auth.pag`	Authorization data base page
`/etc/svc.conf`	Data base service selection and security configuration file

NOTE

Only the superuser and members of the group `authread` can access the `/etc/auth.dir` and `/etc/auth.pag` files.

See Also

chfn(1), chsh(1), finger(1), login(1), shexp(1), yppasswd(1yp), passwd(5yp), edauth(8), vipw(8)
Guide to System Environment Setup
Security Guide for Users and Programmers
"Password Security: A Case History", *ULTRIX Supplementary Documents*, Volume III: System Manager

Name

paste – merge file data

Syntax

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).

In the last form, the `paste` command combines 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.

Options

– Used in place of any file name, to read a line from the standard input. (There is no prompting).

–d*list* Replaces characters of all but last file with nontabs characters (default tab). One or more characters immediately following **–d** replace the default *tab* as the line concatenation character. The list is used circularly, i. e. when exhausted, it is reused. In parallel merging (i. e. no **–s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (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"\\\\"*).
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).

–s Merges 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.

Examples

```
ls | paste -d" " -
```

list directory in one column

```
ls | paste - - - -
```

list directory in four columns

```
paste -s -d"\t\n" file
```

combine pairs of lines into lines

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.

See Also

cut(1), grep(1), pr(1)

Name

pc – Pascal compiler

Syntax

pc [*option*] *name...*

Description

The `pc` command is a Pascal compiler. If given an argument file ending with a .p, it compiles and loads the file into an executable file called, by default, a.out.

A program may be separated into more than one .p file. The `pc` command compiles a number of argument .p files into object files (with the extension .o in place of .p). Object files may then be loaded into an executable a.out file. Exactly one object file must supply a *program* statement to successfully create an executable a.out file. The rest of the files must consist only of declarations which logically nest within the program. References to objects shared between separately compiled files are allowed if the objects are declared in `included` header files, whose names must end with .h. Header files may only be included at the outermost level, and thus declare only globally available objects. To allow functions and procedures to be declared, an *external* directive has been added, whose use is similar to the *forward* directive but restricted to appear only in `.h` files. The *function* and *procedure* bodies may not appear in `.h` files. A binding phase of the compiler checks that declarations are used consistently, to enforce the type checking rules of Pascal. The binder is not as strict as described here, with regard to the rules about external declarations only in .h files and including .h files only at the outermost level.

Object files created by other language processors may be loaded together with object files created by `pc`. The *functions* and *procedures* they define must have been declared in .h files included by all the .p files which call those routines. Calling conventions are as in C, with *var* parameters passed by address.

Options

The following options have the same meaning as in `cc(1)` and `f77(1)`. See `ld(1)` for load-time options.

–c	Suppresses loading and produce .o files from source files.
–g	Produces additional symbol table information for `dbx(1)`.
–w	Suppresses warning messages.
–O	Invokes an object-code improver.
–o *output*	Names the final output file *output* instead of a.out.
–p	Prepares object files for profiling. For further information, see `prof(1)`.
–S	Compiles the named program, and leave the assembler-language output on the corresponding file suffixed .s. No .o file is created.

The following options are peculiar to `pc`.

–C	Compiles code to perform runtime checks, verify `assert` calls, and initialize all variables to zero as in *pi*.
–b	Block buffers the file *output*.
–i*name*	Produces a listing for the specified procedures, functions and `include` files.
–l	Makes a program listing during translation.
–s	Accepts standard Pascal only and non-standard constructs cause warning diagnostics.

Because the **–s** option is usurped by the compiler, it is not possible to pass the strip option to the loader. Thus programs which are to be stripped, must be run through `strip(1)` after they are compiled.

–t *directory*	Uses the given *directory* for compiler temporary files.
–z	Allows execution profiling with `pxp` by generating statement counters, and arranging for the creation of the profile data file pmon.out when the resulting object is executed. The **–z** flag doesn't work for separately compiled files.

Other arguments are taken to be loader option arguments, perhaps libraries of `pc` compatible routines. Certain flags can also be controlled in comments within the program.

Restrictions

The keyword *packed* is recognized but has no effect.

Diagnostics

See `pi(1)` for an explanation of the error message format.

Files

file.p	pascal source files
/usr/lib/pc0	compiler
/lib/f1	code generator
/usr/lib/pc2	runtime integrator (inline expander)
/lib/c2	peephole optimizer
/usr/lib/pc3	separate compilation consistency checker
/usr/lib/pc2.*strings	text of the error messages
/usr/lib/how_pc	basic usage explanation
/usr/lib/libpc.a	intrinsic functions and I/O library
/usr/lib/libm.a	math library
/lib/libc.a	standard library, see intro(3)

See Also

pi(1), pxp(1), pxref(1)
"Berkeley Pascal User's Manual," ULTRIX Supplementary Documents, Vol. II:Programmer

Name

pdx – pascal debugger

Syntax

pdx [**–r**] [*objfile*]

Description

The `pdx` command is a tool for source level debugging and execution of Pascal programs. The *objfile* is an object file produced by the Pascal translator `pi`(1). If no *objfile* is specified, `pdx` looks for a file named ''obj'' in the current directory. The object file contains a symbol table which includes the name of the all the source files translated by `pi` to create it. These files are available for perusal while using the debugger.

If the file .pdxinit exists in the current directory, then the debugger commands in it are executed.

The **–r** option causes the *objfile* to be executed immediately; if it terminates successfully `pdx` exits. Otherwise it reports the reason for termination and offers the user the option of entering the debugger or simply letting `px` continue with a traceback. If **–r** is not specified, `pdx` just prompts and waits for a command.

The commands are:

run [*args*] [< *filename*] [> *filename*]
Start executing *objfile*, passing *args* as command line arguments; < or > can be used to redirect input or output in the usual manner.

trace [**in** *procedure/function*] [**if** *condition*]
trace *source-line-number* [**if** *condition*]
trace *procedure/function* [**in** *procedure/function*] [**if** *condition*]
trace *expression* **at** *source-line-number* [**if** *condition*]
trace *variable* [**in** *procedure/function*] [**if** *condition*]
Have tracing information printed when the program is executed. A number is associated with the command that is used to turn the tracing off (see the **delete** command).

The first argument describes what is to be traced. If it is a *source-line-number*, then the line is printed immediately prior to being executed. Source line numbers in a file other than the current one must be preceded by the name of the file and a colon, for example, ''mumble.p:17''.

If the argument is a procedure or function name then every time it is called, information is printed telling what routine called it, from what source line it was called, and what parameters were passed to it. In addition, its return is noted, and if it's a function then the value it is returning is also printed.

If the argument is an *expression* with an **at** clause then the value of the expression is printed whenever the identified source line is reached.

If the argument is a variable then the name and value of the variable is

printed whenever it changes. Execution is substantially slower during this form of tracing.

If no argument is specified then all source lines are printed before they are executed. Execution is substantially slower during this form of tracing.

The clause "**in** *procedure/function*" restricts tracing information to be printed only while executing inside the given procedure or function.

Condition is a Pascal boolean expression and is evaluated prior to printing the tracing information; if it is false then the information is not printed.

There is no restriction on the amount of information that can be traced.

stop if *condition*
stop at *source-line-number* [**if** *condition*]
stop in *procedure/function* [**if** *condition*]
stop *variable* [**if** *condition*]
Stop execution when the given line is reached, procedure or function called, variable changed, or condition true.

delete *command-number*
The trace or stop corresponding to the given number is removed. The numbers associated with traces and stops are printed by the **status** command.

status [> *filename*]
Print out the currently active **trace** and **stop** commands.

cont Continue execution from where it stopped. This can only be done when the program was stopped by an interrupt or through use of the **stop** command.

step Execute one source line.

next Execute up to the next source line. The difference between this and **step** is that if the line contains a call to a procedure or function the **step** command will stop at the beginning of that block, while the **next** command will not.

print *expression* [, *expression* ...]
Print out the values of the Pascal expressions. Variables declared in an outer block but having the same identifier as one in the current block may be referenced as "*block-name* . *variable*".

whatis *identifier*
Print the declaration of the given identifier.

which *identifier*
Print the full qualification of the given identifier, that is the outer blocks that the identifier is associated with.

assign *variable expression*
Assign the value of the expression to the variable.

call *procedure(parameters)*
Execute the object code associated with the named procedure or function.

help Print out a synopsis of `pdx` commands.

gripe Invokes a mail program to send a message to the person in charge of `pdx`.

where Print out a list of the active procedures and functions and the respective source line where they are called.

source *filename*
Read `pdx` commands from the given *filename*. Especially useful when the *filename* has been created by redirecting a **status** command from an earlier debugging session.

dump [> *filename*]
Print the names and values of all active data.

list [*source-line-number* [, *source-line-number*]]
list *procedure/function*
List the lines in the current source file from the first line number to the second inclusive. As in the editor "$" can be used to refer to the last line. If no lines are specified, the entire file is listed. If the name of a procedure or function is given lines *n-k* to *n+k* are listed where *n* is the first statement in the procedure or function and *k* is small.

file [*filename*]
Change the current source file name to *filename*. If none is specified then the current source file name is printed.

edit [*filename*]
edit *procedure/function-name*
Invoke an editor on *filename* or the current source file if none is specified. If a *procedure* or *function* name is specified, the editor is invoked on the file that contains it. Which editor is invoked by default depends on the installation. The default can be overridden by setting the environment variable EDITOR to the name of the desired editor.

pi Recompile the program and read in the new symbol table information.

sh *command-line*
Pass the command line to the shell for execution. The SHELL environment variable determines which shell is used.

alias *new-command-name old-command-name*
This command makes `pdx` respond to *new-command-name* the way it used to respond to *old-command-name*.

quit Exit `pdx`.

The following commands deal with the program at the *px* instruction level rather than source level. They are not intended for general use.

tracei [*address*] [**if** *cond*]
tracei [*variable*] [**at** *address*] [**if** *cond*]
stopi [*address*] [**if** *cond*]
stopi [**at**] [*address*] [**if** *cond*]
Turn on tracing or set a stop using a *px* machine instruction addresses.

xi *address* [, *address*]
Print the instructions starting at the first *address*. Instructions up to the second *address* are printed.

xd *address* [, *address*]
Print in octal the specified data location(s).

Options

–r Causes `obifile` to be executed immediately. Normally `pdx` prompts and waits for a command.

Restrictions

The `pdx` command does not understand sets, and provides no information about files.

The `whatis` command doesn't quite work for variant records.

Unexpected results occur if a procedure invoked with the **call** command does a non-local goto.

Files

`obj` Pascal object file

`.pdxinit`
`pdx` initialization file

See Also

pi(1), px(1)

Name

pg – file perusal filter for soft-copy terminals

Syntax

pg [*–number*] [**–p** *string*] [**–cefs**] [*+linenumber*] [*+/pattern/*] [files...]

Description

The pg command is a filter that allows the examination of *files* one screenful at a time on a soft-copy terminal. When the file name is designated by a minus (–) and/or NULL argument, the pg command reads from the standard input. Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed.

This command is different from previous paginators because it allows you to back up and review material that has already passed.

In order to determine terminal attributes, pg scans the terminfo(5) data base for the terminal type specified by the environment variable TERM. If TERM is not defined, the terminal is assumed to be a dumb terminal. The pg command takes responses that can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands causing further perusal normally take a preceding *address,* which is 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*>
: Causes one page to be displayed. The address is specified in pages.

(+1) **l**
: Causes pg to simulate scrolling the screen, forward or backward, the number of lines specified when used with a relative address. 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**
: 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*/ — Searches forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i^*pattern*^
i?*pattern*?
Searches backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The circumflex (^) notation is useful for Adds 100 terminals which do not handle the question mark (?) properly.

After searching, pg normally displays 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** — Begins perusing the *i*th next file in the command line. The *i* is an unsigned number. Default value is 1.

i — Begins perusing the *i*th previous file in the command line. *i* is an unsigned number. Default is 1.

*i***w** — Displays another window of text. If *i* is present, sets the window size to *i*.

s *filename* — Saves the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a *<newline>*, even if the *–n* option is specified.

h — Helps by displaying an abbreviated summary of available commands.

q or **Q** — Quits pg.

!*command* — The *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 to 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, because any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then pg acts just like cat(1), except that a header is printed before each file (if there is more than one).

Options

The command line options are:

–number	Specifies 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 is replaced by the current page number when the prompt is issued. The default prompt string is designated by a colon (:).
–c	Homes the cursor and clears the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the terminfo(5) data base.
–e	Causes pg *not* to pause at the end of each file.
–f	Inhibits pg from splitting lines. 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 prevents the splitting of these sequences.
–s	Causes pg to print all messages and prompts in standout mode (usually inverse video).
+linenumber	Starts up at *linenumber*.
+/pattern/	Starts up at the first line containing the regular expression pattern.

Examples

The following example shows how the pg command is used reading system news:

```
news | pg -p "(Page %d):"
```

Notes

While waiting for terminal input, pg responds to **BREAK**, **DEL** and the circumflex (^) by terminating execution. Between prompts, however, these signals interrupt pg command'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.

Restrictions

Terminal tabs must be set every eight positions.

Using pg as a filter with another command changes the terminal I/O options. For example, crypt(1) terminal settings may not be restored correctly.

Files

`/usr/lib/terminfo/*`	Terminal information data base
`/tmp/pg*`	Temporary file when input is from a pipe

See Also

crypt(1), ed(1), grep(1), terminfo(5)

Name

pi – Pascal interpreter code translator

Syntax

pi [*options*] [**–i** *name...*] *name.p*

Description

The `pi` command translates the program in the file name.p leaving interpreter code in the file *obj* in the current directory. The interpreter code can be executed using `px`. The `pix` command performs the functions of `pi` and `px` for 'load and go' Pascal.

Options

The following flags are interpreted by `pi`. The associated options can also be controlled in comments within the program as described in the "Berkeley Pascal User's Manual."

–b Block buffers the file *output*.

–i Enables listing for specified procedures and functions and while processing specified `include` files.

–l Creates a program listing while translating source.

–n Begins each listed `include` file on a new page with a banner line.

–p Suppresses control flow backtrace on error; suppresses statement limit counting.

–s Accepts standard Pascal only; non-standard constructs cause warning diagnostics.

–t Suppresses runtime tests of subrange variables and treat; treats `assert` statements as comments.

–u Runs in card image mode; only the first 72 characters of input lines are used.

–w Suppresses all warning diagnostics.

–z Enables execution profiling with `pxp` by generating statement counters, and arranging for the creation of the profile data file *pmon.out* when the resulting object is executed.

Restrictions

The keyword **packed** is recognized but has no effect.

When **include** files are present, diagnostics relating to the last procedure in one file may appear after the beginning of the listing of the next.

Diagnostics

For a basic explanation type:

```
pi
```

In the diagnostic output of the translator, lines containing syntax errors are listed with a flag indicating the point of error. Diagnostic messages indicate the action which the recovery mechanism took in order to be able to continue parsing. Some diagnostics indicate only that the input is 'malformed.' This occurs if the recovery can find no simple correction to make the input syntactically valid.

Semantic error diagnostics indicate a line in the source text near the point of error. Some errors evoke more than one diagnostic to help pinpoint the error; the follow-up messages begin with an ellipsis '...'.

The first character of each error message indicates its class:

E	Fatal error; no code is generated.
e	Non-fatal error.
w	Warning – a potential problem.
s	Non-standard Pascal construct warning.

If a severe error occurs which inhibits further processing, the translator gives a diagnostic and then 'QUIT'.

Files

file.p	input file
file.i	include file(s)
/usr/lib/pi3.*strings	text of the error messages
/usr/lib/how_pi*	basic usage explanation
obj	interpreter code output

See Also

pix(1), px(1), pxp(1), pxref(1)
"Berkeley Pascal User's Manual," *ULTRIX Supplementary Documents,* Vol. II: Programmer

Name

pick – select messages by content

Syntax

pick [*+folder*] [*msgs*] [**–and ...**] [**–or ...**] [**–not ...**] [**–lbrace ... –rbrace**] [**–cc** *pattern*] [**–date** *pattern*] [**–from** *pattern*] [**–search** *pattern*] [**–subject** *pattern*] [**–to** *pattern*] [**––othercomponent** *pattern*] [**–after** *date*] [**–before** *date*] [**–datefield** *field*] [**–sequence** *name ...*] [**–public**] [**–nopublic**] [**–zero**] [**–nozero**] [**–list**] [**–nolist**] [**–help**]

Description

`Pick` lets you search messages in a folder on a diverse range of search criteria.

You can search the mail headers or the text of some or all of the messages within a folder for the specified criteria. You can use pattern matching or date constraint operations. You can define the messages found as a sequence or you can use the results directly. The following example shows `pick` identifying all the mail in the current folder that was sent by Christine.

```
% pick -from christine
1
3
8
```

The search of the mail header fields that has just been performed is case insensitive. Therefore in the previous example, `pick` would find messages that were from Christine as well as messages that were from christine. However the full `-search` option is not case insensitive. The way in which the `-search` option works is described later in this reference page.

Options

You can search on the contents of any known mail header field by specifying the header field as a `-option`. The mail header fields that `pick` knows about are `To:, cc:, Date:, From:,` or `Subject:`.

If you want to search on some other mail header field, you can do this by using the `--` `component` *pattern* option. If you use this option, you must precede the mail header field with two minus signs. The following example shows `pick` being used to search on the `reply-to:` field.

```
% pick --reply-to Scott
17
```

If you want to search the entire message, and not just the message header, you can do this using the `-search` option. The `-search` option is a modified `grep`(1). As such it gives full regular expression capabilities (see `ed`(1)) within pattern matching.

Pattern matching is performed on a per-line basis. Within the header of the message, each component is treated as one long line, but in the body, each line is separate. Lower-case letters in the search pattern will match either lower or upper case in the message, while upper case will match only upper case.

The following example shows a pick command that will search all messages in the current folder for the word *strawberries*.

```
% pick -search "strawberries"
2
4
```

You can specify a folder other than the current folder using the +foldername option. Also, if you do not want to search all messages within the specified folder, you can specify more than one message or a range of messages using the message numbers. In the following example pick searches messages 10–20 in the +sent folder for messages that were sent to Kafka.

```
% pick +sent 10-20 -to Kafka
pick: no messages match specification
```

You can define a sequence name in which pick can store the details of the messages found. You can manipulate these sequences with any of the MH commands that will take the *msgs* argument.

A sequence name should always begin with a letter and should not contain any non-alphanumeric characters such as punctuation marks. The following example shows the creation of a sequence called Jack.

```
% pick -from Jack -sequence Jack
3 hits
```

Whenever pick processes a -sequence name, it calls the -nolist option. This suppresses the list of messages meeting the search criteria that pick normally generates. Instead pick indicates how many records met the search criteria.

pick normally zeros the sequence before adding to it. This means that any messages that are already in the defined sequence will be lost. If you want to save the messages that are already in the defined sequence, you can do so by specifying the -nozero option. In this case the new messages will be added to the existing messages and numbered accordingly.

If the messages that make up a message sequence are read only, the message sequence that you create will not be accessible to other MH users. If you want to change this you can do so with the -public and -nopublic switches. You can use the -nopublic switch to ensure that the sequence that you create is only accessible to yourself.

In addition to this, you can also combine the output of pick directly with any MH command. In order to do this you need to encase the pick command, and its associated options and arguments, inside single back quotes (‘ ‘). This technique is known as back quoting and is familiar to most ULTRIX users. You should consult your ULTRIX shell Reference Pages for more details. The following example would give you a listing of all the mail sent to you from Jones. See scan(1mh) for details of the listing that would be produced.

```
% scan `pick -from jones`
```

If pick finds that there is no mail from Jones, it will output the illegal character 0. This will cause the MH command, that was going to process the output from the pick operation, to fail gracefully.

Independent of any pattern-matching operations requested, the switches -after *date* or -before *date* may also be used to introduce date/time constraints on all of the messages. By default, the Date: field is consulted. However you can specify an

alternative date field, such as `Delivery-Date:`, by using the `-datefield` *field* option. You must define the `-datefield` before you use the `-after` or `-before` switch it applies to. You must place the arguments to the `-after` or `-before` switches inside double-quotes (" ").

The following example shows `pick` searching for all messages sent to Holloway since 10th June.

```
% pick -to holloway -after "10 Jun 88"
19
```

`Pick` will actually parse the date fields in each of the specified messages and compare them to the date/time specified by the `-after` and `-before` switches. If `-after` is given, then only those messages whose `Date:` field value is chronologically after the date specified will be examined. The `-before` switch specifies the complementary action.

Both the `-after` and `-before` switches take legal RFC 822–style date specifications as arguments. `pick` will assume certain missing fields so that the entire date need not be specified. These fields are (in order of defaulting): timezone, time and timezone, date, date and timezone. All defaults are taken from the current date, time, and timezone.

In addition to RFC 822–style dates, `pick` will also recognize any of the days of the week `Sunday`, `Monday`, and so on), and the special dates `today`, `yesterday`, and `tomorrow`. All days of the week refer to a day in the past. Therefore `pick` interprets `Saturday` as last Saturday and not this Saturday. Finally, in addition to these special specifications, you can also direct `pick` to start its search a number of days ago. The following example searches for messages sent about strawberries in the last ten days.

```
% pick -subject: strawberries -10
1
6
```

The `pick` command also supports complex boolean operations on the searching primitives with the `-and`, `-or`, `-not` , and `-lbrace ... -rbrace` switches. For example,

```
% pick -after yesterday -and -lbrace -from freida  -or -from fear -rbrace
```

identifies messages recently sent by frieda or fear.

The matching primitives take precedence over the `-not` switch, which in turn takes precedence over `-and` which in turn takes precedence over `-or`.

You can override the default precedence with the `-lbrace` and `-rbrace` switches. These act just like opening and closing parentheses in logical expressions.

Restrictions

The sequence name, punctuation and message list must not exceed 1024 characters. In practice, this gives a reasonable limit of approximately 200 non-consecutive messages in a sequence.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine your MH directory
Current–Folder:	To find the default current folder

See Also

ed(1), grep(1), inc(1mh), mark(1mh)

Name

pix – Pascal interpreter and executor

Syntax

pix [**–blnpstuwz**] [**–i** *name...*] *name.p* [*argument...*]

Description

The `pix` command is a 'load and go' version of Pascal which combines the functions of the interpreter code translator `pi` and the executor `px`. It uses `pi` to translate the program in the file name.p and, if there were no fatal errors during translation, causes the resulting interpreter code to be executed by `px` with the specified arguments. A temporary file is used for the object code. The file *obj* is neither created nor destroyed.

Options

–b	Block buffers the output.
–i*name*	Enables the listing for any specified procedures and functions, and while processing any specified include files.
–l	Creates a program listing while translating source.
–n	Begins each listed include file on a new page and with a banner line.
–p	Suppresses control flow backtraces on error.
–s	Accepts standard Pascal only.
–t	Suppresses runtime test of subrange variables.
–u	Runs in card image mode.
–w	Suppresses all warning diagnostics.
–z	Enables execution profiling.

Files

/usr/ucb/pi	Pascal translator
/usr/ucb/px	Pascal executor
/tmp/pix*	temporary
/usr/lib/how_pix	basic explanation

See Also

pi(1), px(1)
"Berkeley Pascal User's Manual," *ULTRIX Supplementary Documents* Vol. II:Programmer

Name

pixie – add profiling code to a program

Syntax

pixie in_prog_name [options]

Description

The **pixie** command reads an executable program, partitions it into basic blocks, and writes an equivalent program containing additional code that counts the execution of each basic block. A basic block is a region of the program that can be entered only at the beginning and exited only at the end. The **pixie** command also generates a file containing the address of each of the basic blocks.

When you run the pixie-generated program, it (provided it terminates normally or via a call to `exit`(2)) generates a file containing the basic block counts. The name of the file is that of the original program with any leading directory names removed and ".Counts" appended. `prof`(1) and `pixstats`(1) can analyze these files and produce a listing of profiling data.

Options

–o *out_prog_name*	Specify a name for the translation. The default is to remove any leading directory names from the in_prog_name and append ".pixie".
–bbaddrs *name*	Specify a name for the file of basic block addresses. Default is to remove any leading directory names from the in_prog_name and append ".Addrs".
–[no]quiet	Controls whether a summary is given of the binary-to-binary translation process. The default is **–noquiet.**
–[no]dwops	Controls translation of double-word load/store instructions so that binaries using these instructions can be run on old processors. The default is **–nodwops** (translate).
–[no]textdata	Controls whether pixie puts the original text into the translated output. This is required to correctly translate programs with data in the text section (for example, f77 format statements in some compiler releases). The default is **–textdata** (include original text).
–[no]idtrace	Disables or enables tracing of instruction and data memory references. **–idtrace** is equivalent to using both **–itrace** and **–dtrace** together. The default is **–noidtrace**
–[no]itrace	Disable or enable tracing of instruction memory references. The default is **–noitrace**
–[no]dtrace	Disable or enable tracing of data memory references. Currently, **–dtrace** requires **–itrace**. The default is **–nodtrace**
–[no]oldtrace	Disable or enable the old memory reference trace format.

The default is **–oldtrace**.

–idtrace_sample *number*
Record only 1 out of every *number* memory reference chunks. (This feature not yet implemented.)

–idtrace_file *number* Specify a UNIX file descriptor number for the trace output file. The default is 19.

Restrictions

The handler function address to the signal system calls is not translated, therefore, programs that receive signals cannot work pixified.

Programs that call **vfork** cannot work pixified because the child process modifies the parent state required for pixie operation. Use **fork** instead.

Pixified code is substantially larger than the original code. Conditional branches that used to fit in the 16-bit branch displacement field may no longer fit, generating a pixie error.

See Also

prof(1), pixstats(1)

Name

pixstats – analyze program execution

Syntax

pixstats program [options]

Description

The **pixstats** command analyzes a program's execution characteristics. To use **pixstats ,** invoke the `pixie`(1) command to translate and instrument the executable object module for the program. Then execute the translation on an appropriate input. This produces a *.Counts* file. You can then use **pixstats** to generate a detailed report on opcode frequencies, interlocks, a mini-profile, and more.

Options

–cycle *ns*	Assume a *ns* cycle time when converting cycle counts to seconds.
–r2010	Use r2010 floating point chip operation times and overlap rules. This is the default.
–disassemble	Disassemble and show the analyzed object code.

Restrictions

The **pixstats** command models execution assuming a perfect memory system. Cache misses, and so on, increase execution above the **pixstats** predictions.

See Also

pixie(1), prof(1)

Name

plot – graphics filters

Syntax

plot [**–T***terminal* [*raster*]] [**–l#**] [**–w#**] [**–c#**]

Description

The `plot` command reads plotting instructions from the standard input and produces plotting instructions for a specified *terminal* on the standard output. For further information see `plot`(5).

Options

The following options are available with the `plot` command.

–T*terminal* Uses the specified terminal name as the terminal type for which plotting instructions are to be generated. If a *terminal* type is not specified, the environment parameter $TERM is used. For more information, see `environ`(7). The *terminal* type can be one of the following:

4020 Tektronix 4020 storage scope.

450 DASI Hyterm 450 terminal (diablo mechanism).

300 DASI 300 or GSI terminal (diablo mechanism).

300S DASI 300S terminal (diablo mechanism).

4014 or tek
Tektronix 4014 or 4015 with Enhanced Graphics Module. (Use 4013 for Tektronix 4014 or Tektronix 4015 without the Enhanced Graphics Module).

4013 Tektronix 4013 Storage scope

aed AED 512 color graphics terminal

bitgraph or bg
BBN bitgraph graphics terminal

crt Any crt capable of running `vi`(1)

dumb dumb terminals without cursor addressing or line printers

grn given a plot file, produces a grn file.

hp7221 Hewlett Packard 7221 Graphics terminal.

hp2648 Hewlett Packard 2648 Graphics terminal.

imagen or ip
Imagen laser printer (default 240 DPI resolution).

lvp16 DEC LVP16 Graphics Plotter.

hp7475a HP 7475A Graphics Plotter.

	ver	Versatec D1200A printer-plotter. This version of `plot` places a scan-converted image in `/usr/tmp/raster` and sends the result directly to the plotter device rather than to the standard output. The optional argument causes a previously scan-converted file *raster* to be sent to the plotter.
	var	Benson Varian printer-plotter.
	vt125	DEC vt125 terminal.
raster		Is a scan-converted temporary file that is sent directly to the plotter. The *raster* file is only specified with the -Tver option.

If *terminal* is either an lvp16 or a hp7475a, you can specify the following options. These options must follow the **–T***terminal* option:

–l#	length of paper window in plotter units (unit scale)
–w#	width of paper window in plotter units (unit scale)
–c#	initial pen carousel to be used

Restrictions

A lockout protection does not exist for `/usr/tmp/raster`.

Files

/usr/bin/t4013
/usr/bin/aedplot
/usr/bin/bgplot
/usr/bin/crtplot
/usr/bin/dumbplot
/usr/bin/gigiplot
/usr/bin/grnplot
/usr/bin/hpplot
/usr/bin/hp7221plot
/usr/bin/implot
/usr/bin/lvp16
/usr/bin/tek
/usr/bin/t450
/usr/bin/t300
/usr/bin/t300s
/usr/bin/vplot
/usr/tmp/raster

See Also

graph(1g), plot(3x), plot(5)

Name

pmerge – pascal file merger

Syntax

pmerge *name.p...*

Description

The `pmerge` command assembles the named Pascal files into a single standard Pascal program. The resulting program is listed on the standard output. It is intended to be used to merge a collection of separately compiled modules so that they can be run through `pi`, or exported to other sites.

Restrictions

Very minimal error checking is done, so incorrect programs will produce unpredictable results. Block comments should be placed after the keyword to which they refer or they are likely to end up in bizarre places.

Files

/usr/tmp/MG*	default temporary files

See Also

pc(1), pi(1),
Auxiliary documentation "Berkeley Pascal User's Manual," *ULTRIX Supplementary Documents* Vol. II:Programmer

pr(1)

Name

pr – print files

Syntax

pr [*options*] [*files*]

Description

The `pr` command prints the named files on the standard output. If *file* is designated by a minus sign (–), or if no files are specified the `pr` command assumes standard input. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space. Lines that do not fit are truncated. However, if the **–s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until `pr` has finished printing.

Options

The following *options* can be used singly or in combination:

–a Prints multi-column output across the page.

–b Prints blank headers.

–d Double-spaces the output.

–e*ck* Expands *input* tabs to character positions *k*+1, 2**k*+1, 3**k*+1,... *n***k*+1. If *k* is 0 or is omitted, tabs are set at every eighth position. Tab characters in the input are expanded into the appropriate number of spaces. The default for *c* (any non-digit character) is the tab character; therefore, if *c* is given, it is treated as the input tab character.

–f Uses form-feed character for new pages. The default is to use a sequence of line-feeds. The **-f** option causes the `pr` command to pause before beginning the first page if the standard output is associated with a terminal.

–h Uses the next argument as the header to be printed instead of the file name.

–i*ck* Replaces white space in *output* by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1,...*n***k*+1. If *k* is 0 or is omitted, tabs are set at every eighth position. The default for *c* (any non-digit character) is the tab character; therefore, if *c* is given, it is treated as the input tab character.

+*k* Begins printing with page *k* (default is 1).

–*k* Produces *k*-column output (default is 1). The **–e** and **–i** options are assumed for multi-column output.

–l*k* Sets the length of a page to *k* lines. The default is 66 lines.

–m Merges and prints all files simultaneously, one per column (overrides the *–k*, and **–a** options).

–n*ck* Numbers lines. The default for **k** is 20. The number occupies the first *k*+1

character positions of each column of normal output or each line of **–m** output. If *c*, which is any non-digit character is given, it is appended to the line number to separate it from whatever follows. The default for *c* is a tab.

–o*k* Offsets each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

–p Pauses before beginning each page if the output is directed to a terminal. The `pr` command rings the bell at the terminal and awaits a carriage return.

–r Suppresses diagnostic reports on failure to open files.

–s*c* Separates columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

–t Suppresses the five-line identifying header and the five-line trailer normally supplied for each page. The **-t** option causes the `pr` command to quit printing after the last line of each file without spacing to the end of the page.

–w*k* Sets the width of a line to *k* character positions. The default is 72 for equal-width multi-column output; otherwise there is no limit.

Examples

Print *file1* and *file2* as a double-spaced, three-column listing with the heading: file list.

```
pr -3dh "file list" file1 file2
```

Write *file1* on *file2*, expanding tabs to columns 10, 19, 28, 37,...:

```
pr -e9 -t <file1>file2
```

Files

/dev/tty* to suspend messages

See Also

cat(1)

prev(1mh)

Name

prev – show the previous message

Syntax

prev [*+foldername*] **[–header] [–noheader] [–showproc** *program*] **[–noshowproc] [–switches for** `showproc`] **[–help]**

Description

The command `prev` displays the previous message in the current folder. You can specify a folder other than the current folder by typing the *+foldername* argument after the command. The command in the following example will display the previous message in the folder +copylog. If you specify a folder, that folder will become the current folder.

```
$ prev +copylog
```

The `prev` command performs a `show` on the previous message in the specified folder. Like `show`, it passes any switches on to the program named by `showproc`, which is called to list the message (see `show`(1mh)). When a message has been displayed by `prev`, it becomes the current message.

The `prev` command is really a link to the `show` program. As a result, if you make a link to `prev` and that link is not called `prev`, your link will act like `show` instead. To avoid this, add a profile-entry for the link to your `MH` profile and add the argument `prev` to the entry.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine your MH directory
Current-Folder:	To find the default current folder
showproc:	Program to show the message

See Also

show(1mh), next(1mh)

Name

print – pr to the line printer

Syntax

print *file...*

Description

The `print` command uses `pr` to print a copy of each named file on the line printer. It is a one line shell script:

```
lpr -p $*
```

See Also

lpr(1), pr(1)

printenv (1)

Name

printenv – display value of a shell variable

Syntax

printenv [*name*]

Description

The `printenv` command prints out the values of the variables in the environment. If a *name* is specified, only its value is printed.

If a *name* is specified and it is not defined in the environment, `printenv` returns exit status 1, else it returns status 0.

See Also

csh(1), environ(7), sh(1)

Name

prmail – print out mail in the post office

Syntax

prmail [*user...*]

Description

The `prmail` command prints the mail which waits for you, or the specified user, in the post office. The mail is not disturbed.

Files

/usr/spool/mail/* post office

See Also

biff(1), binmail(1), from(1), mail(1)

Name

prof – analyze profile data

Syntax

prof [*options*] [*prog_name*] [*pcsampling_data_file* ...]
prof –pixie [**–note** *comment_string*] [*options*] [*prog_name* [*bbaddrs_file* [*bbcounts_file* ...]]]

Description

The `prof` command analyzes one or more data files generated by the compiler's execution-profiling system and produces a listing. Prof can also combine those data files or produce a feedback file that lets the optimizer take into account the program's runtime behavior during a subsequent compilation. Profiling is a three-step process: first compile the program, then execute it, and finally run `prof` to analyze the data.

The compiler system provides two kinds of profiling:

1. *pc-sampling* interrupts the program periodically, recording the value of the program counter.
2. *basic-block counting* divides the program into blocks delimited by labels, jump instructions, and branch instructions. It counts the number of times each block executes. This provides more detailed (line by line) information than pc-sampling.

Using Pc-sampling

To use pc-sampling, compile your program with the option **–p** (strictly speaking, it is sufficient to use this option only when linking the program.) Then run the program, which allocates extra memory to hold the profile data, and (provided the program terminates normally or calls `exit`(2)) records the data in a file at the end of execution.

The environment variable PROFDIR determines the name of the pc-sampling data file and determines whether pc-sampling takes place: if it is not set, the pc-sampling data file is named **mon.out** ; if it is set to the empty string, no profiling occurs; if it is set to a non-empty string, the file is named *string/pid.progname,* where *pid* is the process id of the executing program and *progname* is the program's name, as it appears in argv[0]. The subdirectory *string* must already exist.

After running your program, use `prof` to analyze the pc-sampling data file.

For example:

```
cc -c myprog.c
cc -p -o myprog myprog.o
myprog                      (generates "mon.out")
prof myprog mon.out
```

When you use `prof` for pc-sampling, the program name defaults to **a.out** and the pc-sampling data file name defaults to **mon.out** ; if you specify more than one pc-sampling data file, `prof` reports the sum of the data.

Using Basic-block Counting

To use basic-block counting, compile your program without the option **-p** . Use `pixie`(1) to translate your program into a profiling version and generate a file, whose name ends in .Addrs, containing block addresses. Then run the profiling version, which (assuming the program terminates normally or calls `exit`(2)) will generate a file, whose name ends in .Counts, containing block counts. Then use `prof` with the **-pixie** option to analyze the *bbaddrs* and *bbcounts* files. Notice that you must tell `prof` the name of your original program, not the name of the profiling version.

For example:

```
cc -c myprog.c
cc -o myprog myprog.o
pixie -o myprog.pixie myprog            (generates "myprog.Addrs")
myprog.pixie                        (generates "myprog.Counts")
prof -pixie myprog myprog.Addrs myprog.Counts
```

When you use `prof` with the **-pixie** option, the program name defaults to **a.out** , the *bbaddrs* file name defaults to *program_name*.Addrs, and the *bbcounts* file name defaults to *program_name*.Counts. If you specify more than one *bbcounts* file (never specify more than one *bbaddrs* file), `prof` reports the sum of the data. **-note** *comment_string* If you use this argument, the *comment_string* appears near the beginning of the listing as a comment.

Provided you do not use **-pixie**, `prof` processes mon.out files produced by earlier versions of the compiler system using the obsolete **-p2** or **-p3** options.

Options

For each `prof` option, you need type only enough of the name to distinguish it from the other options (usually the first character is sufficient). Unless otherwise noted, each part of the listing operates only on the set of procedures that results from the combination of the **-exclude** and **-only** options.

If the options you specify would neither produce a listing nor generate a file, `prof` uses **-procedures** plus **-heavy** by default.

-pixie Selects pixie mode, as opposed to pc-sampling mode.

-procedures
Reports time spent per procedure (using data obtained from pc-sampling or basic-block counting; the listing tells which one). For basic-block counting, this option also reports the number of invocations per procedure.

-heavy Reports the most heavily used lines in descending order of use (requires basic-block counting).

-lines Like **-heavy** , but gives the lines in order of occurrence.

-invocations
For each procedure, reports how many times the procedure was invoked from each of its possible callers (requires basic-block counting). For this listing, the **-exclude** and **-only** options apply to callees, but not to callers.

-zero Prints a list of procedures that were never invoked (requires basic-block

counting).

–testcoverage
Reports all lines that never executed (requires basic-block counting).

–feedback *filename*
Produces a file with information that the compiler system can use to decide what parts of the program will benefit most from global optimization and what parts will benefit most from in-line procedure substitution (requires basic-block counting). See `cc`(1).

–merge *filename*
Sums the pc-sampling data files (or, in pixie mode, the *bbcounts* files) and writes the result into a new file with the specified name. The **–only** and **–exclude** options have no effect on the merged data.

–only *procedure_name*
If you use one or more **–only** options, the profile listing includes only the named procedures, rather than the entire program. If any option uses an uppercase O for Only, `prof` uses only the named procedures, rather than the entire program, as the base upon which it calculates percentages.

–exclude *procedure_name*
If you use one or more **–exclude** options, the profiler omits the specified procedure and its descendents from the listing. If any option uses an uppercase E for Exclude, `prof` also omits that procedure from the base upon which it calculates percentages.

–clock *megahertz*
Alters the appropriate parts of the listing to reflect the clock speed of the CPU. If you do not specify *megahertz,* it defaults to 8.0.

–quit *n* Truncates the **–procedures** and **–heavy** listings. It can truncate after *n* lines (if *n* is an integer), after the first entry that represents less than *n* percent of the total (if *n* is followed immediately by a percent character (%)), or after enough entries have been printed to account for *n* percent of the total (if *n* is followed immediately by cum%). For example:

```
-quit 15
```

truncates each part of the listing after 15 lines of text.

```
-quit 15%
```

truncates each part after the first line that represents less than 15 percent of the whole.

```
-quit 15cum%
```

truncates each part after the line that brought the cumulative percentage above 15 percent.

Restrictions

The `prof` command does not yet take into account interactions among floating-point instructions.

Files

crt0.o	normal startup code
mcrt0.o	startup code for pc-sampling
libprof1.a	library for pc-sampling
mon.out	default pc-sampling data file

See Also

as(1), cc(1), pixie(1), profil(2), monitor(3)
Guide to Languages and Programming

prof(1)

Name

prof – profile an object file

Syntax

prof [**–a**] [**–l**] [**–n**] [**–z**] [**–s**] [**–v**] [*low* [*–high*]]] [*file1* [*file2...*]]

Description

The `prof` command interprets the file produced by the `monitor(3)` subroutine. Under default modes, the symbol table in the named object file (a.out default) is read and correlated with the profile file (mon.out default). For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call. If more than one profile file is specified, the output represents the sum of the profiles.

Options

In order for the number of calls to a routine to be tallied, the **–p** option of `cc(1)`, `f77(1)` or `pc(1)` must have been given when the file containing the routine was compiled. This option also arranges for the profile file to be produced automatically.

–a Displays all symbols rather than just external symbols.

–l Displays output by symbol value.

–n Displays output by number of calls.

–s Summary profile file is produced in mon.sum. This is useful only when more than one profile file is specified.

–v Produces graphic output for display by the `plot(1g)` filters. When plotting, the numbers *low* and *high,* by default 0 and 100, may be given, which causes a selected percentage of the profile to be plotted with accordingly higher resolution.

–z Routines having zero usage, as indicated by call counts and accumulated time, are printed in the output.

Restrictions

Beware of quantization errors.

The `f77` command causes confusion because the entry points are at the bottom of subroutines and functions.

Files

mon.out for profile
a.out for namelist
mon.sum for summary profile

See Also

cc(1), plot(1g), profil(2), monitor(3)

prompter(1mh)

Name

prompter – prompting editor front-end

Syntax

prompter [–ddif] [–erase *chr*] **[–kill** *chr*] **[–prepend | –noprepend] [–rapid | –norapid]** *file* [–help]

Description

The `prompter` editor is a rudimentary editor provided by `comp, dist, forw,` or `repl.` You do not need to specify this editor. It is automatically called by the above commands. You can change this editor to an editor of your choice, by inserting the following line in your `.mh_profile` file.

```
editor:editorname
```

Any ULTRIX editor can be specified in your `.mh_profile.`

You can also specify an alternative editor each time that you use `comp, dist, forw,` or `repl,` by using the `-editor` *editorname* option. This is useful if you have special requirements for a message. You can use an editor other than the one you normally use. The following example shows how you can compose a message using `vi` as the editor:

```
$ comp –editor vi
```

The `prompter` editor allows rapid composition of messages. It is particularly useful to network and low-speed (less than 2400 baud) users of `MH.` The `prompter` editor is an `MH` program. It can have its own profile entry with switches, but it is not invoked directly. The commands `comp, dist, forw,` and `repl` invoke `prompter` as an editor, either when invoked with `-editor prompter,` or by the profile entry `Editor: prompter,` or when given the command `edit prompter` when prompted with `What now?.`

For each empty component `prompter` finds in the draft, you are prompted for a response; a <RETURN> will cause the whole component to be left out. Otherwise, a back-slash (`\`) preceding a <RETURN> will continue the response on the next line, allowing for multiline components. Continuation lines `must` begin with a space or tab.

Each non-empty component is copied to the draft and displayed on the terminal.

The start of the message body is indicated by a blank line or a line of dashes. If the body is non-empty, the prompt, which is not written to the file, is

```
--------Enter additional text
```

or (if `-prepend` was given)

```
--------Enter initial text
```

Typing of the message body is terminated with an end–of–file (usually CTRL–D). At this point control is returned to the calling program, where you are asked `What now?.` See `whatnow`(1mh) for the valid options to this query.

An interrupt (usually CTRL-C) during component typing will abort `prompter` and the MH command that invoked it. An interrupt during message-body typing is equivalent to CTRL-D, for historical reasons. This means that `prompter` should finish up and exit.

The `prompter` editor uses `stdio`(3s), therefore do not edit files with nulls in them.

Options

–ddif Prepare a DDIF document for mailing. The **–ddif** option will accept specification of either a DDIF(5) or DOTS(5) file. If the DDIF(5) file contains external references to other files, these are packed together in a single DOTS(5) encoding as a part of the preparations for packaging the document for mailing.

–prepend Adds text to the beginning of the message body and have the rest of the body follow. This is useful for the `forw` command.

–rapid Causes the text to not be displayed on your terminal if the draft already contains text in the message-body, This is useful for low-speed terminals.

–erase *char* Specifies the line-editing characters where *chr* may be a character or `\nnn`, where `nnn` is the octal value for the character.

–kill *char* Specifies the line-editing characters where *chr* may be a character or `\nnn`, where `nnn` is the octal value for the character.

The first non-flag argument to `prompter` is taken as the name of the draft *file*, and subsequent non-flag arguments are ignored. `Repl` invokes editors with two file arguments:

The default settings for `prompter` are: `-prepend` and `-norapid`

Examples

The following is an example using the -ddif switch, plus **send draft:**

```
% prompter -ddif; send draft
To: <address>
cc: <other recipients>
Subject: <title of the message>
DDIF/DOTS file : <name of the DDIF or DOTS file>
```

Files

$HOME/.mh_profile	The user profile
/tmp/prompter*	Temporary copy of message

Profile Components

prompter–next:	To name the editor to be used on exit from prompter
Msg–Protect:	To set mode when creating a new draft

See Also

capsar(1), comp(1mh), dist(1mh), forw(1mh), repl(1mh), whatnow(1mh), stdio(3s), DDIF(5), DOTS(5), mh_profile(5mh)

Name

prs – display information from an SCCS file

Syntax

prs [**–d**[*dataspec*]] [**–r**[*SID*]] [**–e**] [**–l**] [**–a**] *files*

Description

The `prs` command prints on the standard output all or parts of an SCCS file in a user supplied format. For further information, see `sccsfile`(5). 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.

Options

All the described keyletter arguments apply independently to each named file:

–a	Displays printing of information for both removed and existing deltas. For example, removed delta type = *R*, and existing delta type = *D*. For further information, see `rmdel`(1). If the **–a** keyletter is not specified, information is provided only for existing deltas.
–d[*dataspec*]	Displays information specified by *dataspec*. The *dataspec* is a string consisting of SCCS file *data keywords* (see **DATA KEYWORDS**) interspersed with optional user-supplied text.
–e	Displays information for all deltas created earlier than and including the delta designated by the **–r** keyletter.
–l	Displays information for all deltas created later than and including the delta designated by the **–r** keyletter.
–r[*SID*]	Indicates delta version number. If no SID is specified, the SID of the most recently created delta is assumed.

Data Keywords

Data keywords specify which parts of an SCCS file are to be retrieved and printed on the standard output. All parts of an SCCS file have an associated data keyword. For further information, see `sccsfile`(5). There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by `prs` consists of the user supplied text and the 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.

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::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 following is an example of a `prs` command line and what it produces on the standard output:

```
prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file
```

Users and/or user IDs for s.file are:
xyz
131
abc

This is another example and its output.

```
prs -d"Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file
```

Newest delta for pgm main.c: 3.7 Created 77/12/1 By case

This is an example of a *special case:*

```
prs s.file
```

This is what it may produce 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.

Diagnostics

See `sccshelp(1)` for explanations.

Files

/tmp/pr?????

See Also

admin(1), delta(1), get(1), help(1), sccs(1), sccsfile(5)
Guide to the Source Code Control System

ps(1)

Name

ps – print process status statistics

Syntax

ps [*options*] [*namelist*] [*core*]

Description

The `ps` command prints information about processes. Without the **–a** option, only your processes are candidates to be printed by `ps`. Specifying the **–a** option causes other users' processes to be printed, and specifying the **–x** option includes processes without control terminals.

You must use the **–k** option and specify both *namelist* and *core* files when looking at a crash dump. The optional *namelist* argument indicates that information is to be gathered using the specified system *namelist* file. (The *namelist* file is a kernel image file). If *namelist* is not specified, `ps` uses `/vmunix`. The *core* argument indicates that information is to be gathered using the specified corefile. If *core* is not specified, `ps` uses `/dev/mem`.

Options

–# Represents any given process number and must be the last option given. When used in combinations, this option overrides **–a** and is overridden by **–tx**.

–C Causes the **%CPU** field of the display to reflect the absolute percentage of cpu used by the process during its resident time for scheduling.

–S Causes the **TIME** field of the display to reflect the amount of user+system time spent by a process and its children.

–a Displays information for processes executed from all users' terminals. The default is to show processes executed from your terminal only. When used in combinations, this option is overridden by **–#**.

–c Displays the command names as stored internally in the system for accounting purposes instead of the command arguments, which are kept in the process address space. This display is more reliable, if less informative, because a process is free to destroy the latter information.

–e Displays the environment as well as the command arguments.

–g Displays all processes within the process group. Without this option, `ps` prints only ''interesting'' processes. Processes are considered uninteresting if they are process group leaders; top-level command interpreters and processes waiting for users to login on free terminals are therefore normally not shown.

–k Uses the *core* file in place of `/dev/kmem` and `/dev/mem`. If the **–k** option is used but no *core* file is specified, `ps` uses `/dev/mem`.

–l Displays information in long format, showing the fields PPID, CP, PRI, NI, ADDR, SIZE, RSS, and WCHAN as described under Output Fields.

–s Adds the size SSIZ of the kernel stack of each process to the basic

output format for use by system maintainers.

–t*x* Displays information for specified terminal only. Restricts output to processes whose controlling terminal is *x*. The argument *x* should be specified as printed by `ps`; for example, *t3* for tty3, *tco* for console, *td0* for ttyd0, *t?* for processes with no terminal, *t* for processes at the current tty, and so forth, are proper specifications for *x*. This option must be the last one given. When used in combinations, this option overrides **–#** and **–a**.

–u Displays user-oriented output, which includes fields USER, %CPU, and %MEM, SIZE. It also displays SZ and RSS, which are computed differently than for the **–l** and **–v** options. The SZ field is computed as SIZE + TSIZ (virtual size plus size of text). The TRS field is computed as RSS + (TRS/xccount) where xccount is the number of processes currently sharing the text.

–v Displays process system time and user time in addition to cumulative time. This display includes fields RE, SL, PAGEIN, SIZE, RSS, LIM, TSIZ, TRS, %CPU and %MEM, described under Output Fields.

–w Produces 132-column rather than 80 column output. If repeated, as **ww**, produces arbitrarily wide output. This information is used to decide how much to print for long commands.

–x Displays information for all processes, including those not executed from terminals.

The `ps` command ignores all options not mentioned in the reference page.

Restrictions

Information on processes can change while `ps` is running. The picture it gives is a snapshot taken at a given time.

Output Fields

All output formats include, for each process:

PID The process identification (PID) number

TT Control terminal of the process

TIME Includes both user and system time

STAT The state of the process, given by a sequence of four letters, for example, RWNA.

The first letter indicates the run status of the process:

R Running processes

T Stopped processes

P Processes in page wait

D Processes in disk (or other short-term) waits

S Processes sleeping for less than about 20 seconds

I Idle processes (sleeping longer than about 20 seconds)

The second letter indicates whether a process is swapped out:

W Processes that are swapped out

Z Processes that are killed but not yet removed

(blank)
Processes that are in core

> Processes that have specified a soft limit on memory requirements and are exceeding that limit. Such a process is not swapped.

The third letter indicates whether a process is running with altered CPU scheduling priority, using `nice`(1).

N The process priority is reduced

< The process priority has been artificially raised

(blank)
Processes running without special treatment

The fourth letter indicates any special treatment of the process for virtual memory. The possibilities are:

A Stands for VA_ANOM. Typically represents a `lisp`(1) process making disk usage more efficient by removing gaps caused by deletes and collecting the remaining data.

S Stands for VA_SEQL. Typical of large image processing programs that are using virtual memory to sequentially address voluminous data.

(blank)
Stands for VA_NORM.

Fields that are not common to all output formats:

USER Name of the owner of the process

%CPU CPU utilization of the process. This is a decaying average over a minute or less of previous (real) time. Because the time base over which this is computed varies since processes may be very young, it is possible for the sum of all %CPU fields to exceed 200%.

NICE (or NI) Process scheduling increment. For further information, see `setpriority`(2).

SIZE (or SZ) Virtual size of the process (in 1024-byte units)

RSS Real memory (resident set) size of the process (in 1024-byte units)

LIM Soft limit on memory used, specified via a call to `getrlimit`(2). If no limit has been specified, then shown as *xx*

TSIZ Size of text (shared program) image

TRS Size of resident (real memory) set of text

%MEM Percentage of real memory used by this process.

RE Residency time of the process (seconds in core)

SL Sleep time of the process (seconds blocked)

PAGEIN Number of disk I/O operations resulting from references by the process to pages not loaded in core

UID Numerical user identification number of process owner

PPID Numerical identification number of parent of process

CP Short-term CPU utilization factor used in scheduling

PRI Process priority (nonpositive when in noninterruptible wait)

ADDR Swap address of the process or page frame of the beginning of the user page table entries

WCHAN Event for which process is waiting (an address in the system), with the initial part of the address trimmed off. For example, 80004000 prints as 4000.

F Flags associated with process as in `< sys/proc.h >`:

SLOAD	00000001	Process is resident in memory
SSYS	00000002	System process: swapper, pager, idle (RISC only), trusted path daemon
SLOCK	00000004	Process is being swapped out
SSWAP	00000008	Process requested swapout for page table growth
STRC	00000010	Traced
SWTED	00000020	Used in tracing
SULOCK	00000040	Locked in by plock(2)
SPAGE	00000080	Waiting for page-in to complete
SKEEP	00000100	Protected from swapout while tranferring resources to another process
SOMASK	00000200	Used by sigpause(2)
SWEXIT	00000400	Exiting
SPHYSIO	00000800	Protected from swapout while doing physical I/O
SVFORK	00001000	Process resulted from a vfork(2) that is not yet complete
SVFDONE	00002000	Parent has received resources returned by vfork(2) child
SNOVM	00004000	Process has no virtual memory, as it is a parent in the context of vfork(2)
SPAGI	00008000	Process is demand paging data pages from its text gnode.
SSEQL	00010000	Process has advised of sequential memory access
SUANOM	00020000	Process has advised of random memory access
SXCTDAT	00080000	Process has indicated intent to execute data or stack (RISC only)
SNOCLDSTP	00100000	POSIX environment: no SIGCLD generated when children stop (formerly named SOUSIG)
SOWEUPC	00200000	Process is owed a profiling tick
SSEL	00400000	Used by select(2)
SLOGIN	00800000	A login process
SPTECHG	01000000	The pte's for the process have changed
SLKDONE	04000000	System V file lock applied

SFIXADE	08000000	Fix-up of unaligned accesses is attempted (RISC only)
SEXECDN	10000000	Process has done an execve(2)
SIDLEP	20000000	The idle process (RISC only)

A process that has a parent and has exited, but for which the parent has not yet waited, is marked <defunct>. A process that is blocked trying to exit is marked <exiting>; the `ps` command makes an educated guess as to the file name and arguments given when the process was created by examining memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

Files

`/vmunix`	System namelist
`/dev/kmem`	Kernel memory
`/dev/mem`	User process info
`/dev/drum`	Swap device
`/vmcore`	Core file
`/dev`	Searched to find swap device and tty names

See Also

kill(1), w(1), dump(5)

Name

ptx – create permuted index

Syntax

ptx [*option...*] [*input* [*output*]]

Description

The `ptx` command generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. The `ptx` command produces output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx may be an `nroff` or `troff` macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as fits around the keyword when it is printed at the middle of the page. The `tail` and `head` commands, at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

Options

The following options can be applied:

–b *break*	Use the characters in the *break* file as separators. In any case, tab, new line, and space characters are always used as break characters.
–f	Folds upper and lower case letters for sorting.
–g *n*	Uses specified number as interfield gap. The default gap is 3 characters.
–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.
–o *only*	Use words listed only in the *only* file.
–r	Uses leading nonblanks as reference identifiers. Attach that identifier as a 5th field on each output line.
–t	Prepares the output for the phototypesetter. The default line length is 100 characters.
–w *n*	Use the next argument, *n*, as the width of the output line. The default line length is 72 characters.

Restrictions

Line length counts do not account for overstriking or proportional spacing.

Files

/usr/bin/sort
/usr/lib/eign

Name

pwd – print working directory

Syntax

pwd

Description

The `pwd` command prints the pathname of the working (current) directory.

Restrictions

In `csh`(1) the command *dirs* is always faster (although it can give a different answer in the rare case that the current directory or a containing directory was moved after the shell descended into it).

See Also

cd(1), csh(1), getwd(3)

Name

px – Pascal code executor

Syntax

px [*obj* [*argument...*]]

Description

The `px` command interprets the abstract machine code generated by `pi`. The first argument is the file to be interpreted, and defaults to *obj* ; remaining arguments are available to the Pascal program using the built-ins *argv* and *argc*. The `px` command is also invoked by `pix` when running 'load and go'.

If the program terminates abnormally an error message and a control flow backtrace are printed. The number of statements executed and total execution time are printed after normal termination. The **p** option of `pi` suppresses all of this except the message indicating the cause of abnormal termination.

Restrictions

Post-mortem traceback is not limited; infinite recursion leads to almost infinite traceback.

Diagnostics

Most run-time error messages are self-explanatory. Some of the more unusual ones are:

Reference to an inactive file
: A file other than *input* or *output* was used before a call to *reset* or *rewrite*.

Statement count limit exceeded
: The limit of 500,000 executed statements (which prevents excessive looping or recursion) has been exceeded.

Bad data found on integer read
Bad data found on real read
: Usually, non-numeric input was found for a number. For reals, Pascal requires digits before and after the decimal point so that numbers like '.1' or '21.' evoke the second diagnostic.

panic: *Some message*
: Indicates a internal inconsistency detected in `px` probably due to a Pascal system bug.

Files

obj	default object file
pmon.out	profile data file

See Also

pi(1), pix(1)
"Berkeley Pascal User's Manual," *ULTRIX Supplementary Documents*, Vol. II:Programmer

Name

pxp – Pascal execution profiler

Syntax

pxp [–acdefjnstuw_] [–23456789] [–z[*name...*]] *name.p*

Description

The `pxp` command can be used to obtain execution profiles of Pascal programs or as a pretty-printer. To produce an execution profile all that is necessary is to translate the program specifying the **z** option to `pi` or `pix`, to execute the program, and to then issue the command

```
pxp -z name.p
```

A reformatted listing is output if none of the **c**, **t**, or **z** options are specified; thus

```
pxp old.p > new.p
```

places a pretty-printed version of the program in old.p in the file new.p.

Options

The use of the following options of `pxp` is discussed in sections 2.6, 5.4, 5.5 and 5.10 of the *Berkeley Pascal User's Manual*.

–_ Underscores all keywords.

–*d* Uses the specified number (*–d*) as the indentation unit. The default is 4.

–a Displays all procedures (even those not executed).

–c Uses the `core` file in generating the profiling data.

–d Displays all declaration parts.

–e Eliminates `include` directives when reformatting a file. The `include` is replaced by the reformatted contents of the specified file.

–f Displays all parenthesized expression.

–j Left justifies all procedures and functions.

–n Begins a new page for each included file. In profiles, print a blank line at the top of the page.

–s Strips comments from the input text.

–t Prints a table summarizing `procedure` and `function` call counts.

–u Generates the output in card image format, using only the first 72 characters of input lines.

–w Suppresses all warning diagnostics.

–z Generate an execution profile for the specified modules (next arguments). If no *name* s, are given the profile is of the entire program. If a list of names is given, then only any specified **procedures** or **functions** and the contents of any specified **include** files appear in the profile.

Restrictions

Does not place multiple statements per line.

Diagnostics

For a basic explanation of the `pxp` command, type:

```
pxp
```

Error diagnostics include 'No profile data in file' with the **c** option if the **z** option was not enabled to `pi`; 'Not a Pascal system core file' if the core is not from a `px` execution; 'Program and count data do not correspond' if the program was changed after compilation, before profiling; or if the wrong program is specified.

Files

name.p	input file
name.i	include file(s)
pmon.out	profile data
core	profile data source with **–c**
/usr/lib/how_pxp	information on basic usage

See Also

pi(1), px(1)
"Berkeley Pascal User's Manual," *ULTRIX Supplementary Documents* Vol. II: Programmer

Name

pxref – Pascal cross-reference program

Syntax

pxref [–] *name*

Description

The `pxref` command makes a line numbered listing and a cross-reference of identifier usage for the program in *name*. The keywords `goto` and `label` are treated as identifiers for the purpose of the cross-reference. The `include` directives are not processed, but cause the placement of an entry indexed by '#include' in the cross-reference.

Options

– Optional argument that suppresses the line numbered listing.

Restrictions

Identifiers are trimmed to 10 characters.

See Also

"Berkeley Pascal User's Manual," *ULTRIX Supplementary Documents* Vol II: Programmer

quota(1)

Name

quota – display disk usage and limits

Syntax

quota [**–qv**] [*user*]

Description

The `quota` command displays users' disk usage and limits. Only the super-user may use the optional *user* argument to view the limits of users other than himself.

The `quota` command reports only on file systems which have disk quotas. If `quota` exits with a non-zero status, one or more file systems are over quota.

Options

–q Prints a message that contains information only on file systems where usage is over quota

–v Displays users quotas on file systems where no storage is allocated.

See Also

quota(2), edquota(8), quotaon(8),
"Disk Quotas in a UNIX Environment", *ULTRIX Supplementary Documents* Vol. III: System Manager

Name

ranlib – convert archives to random libraries

Syntax

ranlib *archive...*

Description

The `ranlib` command converts each *archive* to a form which the loader can load more rapidly. The **ranlib** command uses `ar`(1) to reconstruct the archive, so that sufficient temporary file space must be available in the file system which contains the current directory.

See Also

ar(1), ld(1), lorder(1)

Name

ranlib – convert archives to random libraries

Syntax

ranlib *archive...*

Description

The `ranlib` command converts each *archive* to a form which the loader can load more rapidly. The `ranlib` command does this by adding a table of contents called _.SYMDEF to the beginning of the archive. the `ranlib` command uses `ar`(1) to reconstruct the archive, so that sufficient temporary file space must be available in the file system which contains the current directory.

Restrictions

Because generation of a library by `ar` and randomization of the library by `ranlib` are separate processes, phase errors are possible. The loader, `ld`, warns when the modification date of a library is more recent than the creation date of its dictionary; but this means that you get the warning even if you only copy the library.

See Also

ar(1), ld(1), lorder(1)

Name

rcp – remote file copy

Syntax

rcp [**–p**] *file1 file2*
rcp [**–r**] [**–p**] *file... directory*

Description

The `rcp` command copies files between machines. Each *file* or *directory* argument is either a remote file name of the form rhost:path, or a local file name. Local file names do not contain colons (:) or backslashes (\) before colons.

Note that the `rcp` command refuses to copy a file onto itself.

If *path* is not a full path name, it is interpreted relative to your login directory on *rhost*. To ensure that the metacharacters are interpreted remotely, a remote host's *path* can be quoted by either using a backslash (\) before a single character, or enclosing character strings in double (") or single (') quotes.

The `rcp` command does not prompt for passwords; your current local user name must exist on *rhost* and allow remote command execution via `rsh`(1c).

The `rcp` command handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form rname@rhost to use *rname* rather than the current user name on the remote host. The following example shows how to copy the file foo from user1@mach1 to user2@mach2:

```
$ rcp user1@mach1:foo  user2@mach2:foo
```

Note that the file .rhosts on mach2 in user2's account must include an entry for mach1 user1. Also note that it may be necessary for the person implementing the `rcp` command to be listed in the .rhosts file for mach1 user1.

By default, the mode and owner of *file2* are preserved if *file2* already exists. Otherwise, the mode of the source file modified by `umask`(2) on the destination host is used.

Options

–p Preserves the modification times and modes of the source files in its copies, ignoring the `umask`.

–r Copies files in all subdirectories recursively, if the file to be copied is a directory. In this case the destination must be a directory.

Restrictions

The `rcp` command is confused by output generated by commands in a .login, .profile, or .cshrc file on the remote host. In particular, 'where are you?' and 'rcp: protocol screwup' are messages which can result if output is generated by any of the startup files.

See Also

ftp(1c), rlogin(1c), rsh(1c)

Name

rcvstore – incorporate new mail asynchronously

Syntax

rcvstore [+folder] [–create] [–nocreate] [–sequence *name*] **[–public] [–nopublic] [–zero] [–nozero] [–help]**

Description

The command `rcvstore` incorporates a message from the standard input into an `MH` folder. If `+folder` is not specified, the folder named `inbox` in your `MH` directory will be used instead. The new message being incorporated is assigned the next highest number in the folder.

If the specified (or default) folder does not exist, then it will be created if the `-create` option is specified; otherwise `rcvstore` will exit.

If your profile contains a `Msg-Protect:nnn` entry, it will be used as the protection on the newly created messages; otherwise the `MH` default of 0600 will be used. During all operations on messages, this initially assigned protection will be preserved for each message, so `chmod` may be used to set a protection on an individual message, and its protection will be preserved thereafter.

The `rcvstore` command will incorporate anything except zero length messages into your MH folder.

If the profile entry `Unseen-Sequence` is present and non–empty, then `rcvstore` will add the newly incorporated message to each sequence named by the profile entry. This is similar to the `Previous-Sequence` profile entry supported by all `MH` commands which take `msgs` or `msg` arguments. Note that `rcvstore` will not zero each sequence prior to adding messages.

Furthermore, the incoming messages may be added to user-defined sequences as they arrive by appropriate use of the `-sequence` option. As with `pick`, the `-zero` and `-nozero` switches can also be used to zero old sequences. Similarly, the `-public` and `-nopublic` switches may be used to force additions to public and private sequences.

The defaults for this command are:

+folder defaults to `inbox`
`-create`
`-nopublic` if the folder is read–only, `-public` otherwise
`-nozero`

Files

$HOME/.mh_profile The user profile

Profile Components

Path:	To determine your MH directory
Folder-Protect:	To set mode when creating a new folder
Msg-Protect:	To set mode when creating a new message
Unseen-Sequence:	To name sequences denoting unseen messages

See Also

chmod(1), inc(1mh), pick(1mh), mh-mail(5mh)

Name

rdist – remote file distribution program

Syntax

rdist [**–nqbRhivwy**] [**–f** *distfile*] [**–d var**=*value*] [**–m** *host*] [*name ...*]

rdist [**–nqbRhivwy**] [**-c** *name ...* **[login@]***host***[:dest]**

Description

The `rdist` program maintains identical copies of files over multiple hosts. It preserves the owner, group, mode, and mtime of files if possible and can update programs that are executing. `rdist` reads commands from *distfile* to direct the updating of files and/or directories. If *distfile* is '–', the standard input is used. If no **–f** option is present, the program looks first for 'distfile', then 'Distfile' to use as the input. If no names are specified on the command line, `rdist` will update all of the files and directories listed in *distfile*. Otherwise, the argument is taken to be the name of a file to be updated or the label of a command to execute. If label and file names conflict, it is assumed to be a label. These may be used together to update specific files using specific commands.

Options

–c Forces `rdist` to interpret the remaining arguments as a small *distfile*. The equivalent *distfile* is as follows.

```
( name ... ) -> [login@]host
        install  [dest] ;
```

–d Defines **var** to have *value*. The **–d** option is used to define or override variable definitions in the *distfile*. *Value* can be the empty string, one name, or a list of names surrounded by parentheses and separated by tabs and/or spaces.

–m Limit which machines are to be updated. Multiple **-m** arguments can be given to limit updates to a subset of the hosts listed the *distfile*.

–n Print the commands without executing them. This option is useful for debugging *distfile*.

–q Quiet mode. Files that are being modified are normally printed on standard output. The **–q** option suppresses this.

–R Remove extraneous files. If a directory is being updated, any files that exist on the remote host that do not exist in the master directory are removed. This is useful for maintaining truly identical copies of directories.

–h Follow symbolic links. Copy the file that the link points to rather than the link itself.

–i Ignore unresolved links. *Rdist* will normally try to maintain the link structure of files being transferred and warn the user if all the links cannot be found.

–v Verify that the files are up to date on all the hosts. Any files that are out of date will be displayed but no files will be changed nor any mail sent.

–w Whole mode. The whole file name is appended to the destination directory name. Normally, only the last component of a name is used when renaming files. This will preserve the directory structure of the files being copied instead of flattening the directory structure. For example, renaming a list of files such as (dir1/f1 dir2/f2) to dir3 would create files dir3/dir1/f1 and dir3/dir2/f2 instead of dir3/f1 and dir3/f2.

–y Younger mode. Files are normally updated if their *mtime* and *size* (see `stat`(2)) disagree. The **–y** option causes *rdist* not to update files that are younger than the master copy. This can be used to prevent newer copies on other hosts from being replaced. A warning message is printed for files which are newer than the master copy.

–b Binary comparison. Perform a binary comparison and update files if they differ rather than comparing dates and sizes.

Distfile contains a sequence of entries that specify the files to be copied, the destination hosts, and what operations to perform to do the updating. Each entry has one of the following formats.

<variable name> '=' <name list>
[label:] <source list> '–>' <destination list> <command list>
[label:] <source list> '::' <time_stamp file> <command list>

The first format is used for defining variables. The second format is used for distributing files to other hosts. The third format is used for making lists of files that have been changed since some given date. The *source list* specifies a list of files and/or directories on the local host which are to be used as the master copy for distribution. The *destination list* is the list of hosts to which these files are to be copied. Each file in the *source list* is added to a list of changes if the file is out of date on the host which is being updated (second format) or the file is newer than the time stamp file (third format).

Labels are optional. They are used to identify a command for partial updates.

Newlines, tabs, and blanks are only used as separators and are otherwise ignored. Comments begin with a sharp sign (#) and end with a newline.

Variables to be expanded begin with dollar sign ($) followed by one character or a name enclosed in curly braces (see the examples at the end).

The *source list* and *destination list* have the following format:

<name>

or

'(' <zero or more names separated by white-space> ')'

The shell meta-characters [,], {, }, *, and ? are recognized and expanded (on the local host only) in the same way as `csh`. They can be escaped with a backslash (). The tilde character (~) is also expanded in the same way as `csh`, but is expanded separately on the local and destination hosts. When the **–w** option is used with a file name that begins with tilde (~), everything except the home directory is appended to the destination name. File names which do not begin with / or ~ use the destination user's home directory as the root directory for the rest of the file name.

The command list consists of zero or more commands of the following format.

'install' <options> opt_dest_name ';'
'notify' <name list>';'
'except' <name list>';'
'except_pat' <pattern list>';'
'special' <name list>string ';'

The *install* command is used to copy out of date files and/or directories. Each source file is copied to each host in the destination list. Directories are recursively copied in the same way. *opt_dest_name* is an optional parameter to rename files. If no **install** command appears in the command list or the destination name is not specified, the source file name is used. Directories in the path name will be created if they do not exist on the remote host. To help prevent disasters, a non-empty directory on a target host will never be replaced with a regular file or a symbolic link. However, under the **–R** option a non-empty directory will be removed if the corresponding filename is completely absent on the master host. The *options* are **–R, –h, –i, –v, –w, –y,** and **–b** and have the same semantics as options on the command line except they only apply to the files in the source list. The login name used on the destination host is the same as the local host unless the destination name is of the format login@host.

The **notify** command is used to mail the list of files updated (and any errors that may have occurred) to the listed names. If no at sign (@) appears in the name, the destination host is appended to the name (for example, name1@host, name2@host, ...).

The **except** command is used to update all of the files in the source list **except** for the files listed in *name list*. This is usually used to copy everything in a directory except certain files.

The **except_pat** command is like the **except** command except that *pattern list* is a list of regular expressions (see ed(1) for details). If one of the patterns matches some string within a file name, that file will be ignored. Note that since \e is a quote character, it must be doubled to become part of the regular expression. Variables are expanded in *pattern list* but not shell file pattern matching characters. To include a dollar sign ($), it must be escaped with \e.

The *special* command is used to specify sh commands that are to be executed on the remote host after the file in *name list* is updated or installed. If the *name list* is omitted then the shell commands will be executed for every file updated or installed. The shell variable FILE is set to the current filename before executing the commands in *string*. *String* starts and ends with double quotes (") and can cross multiple lines in *distfile*. Multiple commands to the shell should be separated by semi-colons (;). Commands are executed in the user's home directory on the host being updated. The **special** command can be used to rebuild private databases, etc. after a program has been updated.

The following is a small example.

```
HOSTS = ( matisse root@arpa)

FILES = ( /bin /lib /usr/bin /usr/games
        /usr/include/{*.h,{stand,sys,vax*,pascal,machine}/*.h}
        /usr/lib /usr/man/man? /usr/ucb /usr/local/rdist )
```

```
EXLIB = ( Mail.rc aliases aliases.dir aliases.pag crontab dshrc
        sendmail.cf sendmail.fc sendmail.hf sendmail.st uucp vfont )

${FILES} -> ${HOSTS}
        install -R ;
        except /usr/lib/${EXLIB} ;
        except /usr/games/lib ;
        special /usr/lib/sendmail "/usr/lib/sendmail -bz" ;

srcs:
/usr/src/bin -> arpa
        except_pat ( \\o\$ /SCCS\$ ) ;

IMAGEN = (ips dviimp catdvi)

imagen:
/usr/local/${IMAGEN} -> arpa
        install /usr/local/lib ;
        notify ralph ;

${FILES} :: stamp.cory
        notify root@cory ;
```

Restrictions

Source files must reside on the local host where `rdist` is executed.

There is no easy way to have a **special** command executed after all files in a directory have been updated.

Variable expansion only works for *name lists;* there should be a general macro facility.

`rdist` aborts on files which have a negative mtime (before Jan 1, 1970).

Diagnostics

A complaint about mismatch of `rdist` version numbers may really stem from some problem with starting your shell (that is, you are in too many groups).

Files

distfile	input command file
/tmp/rdist*	temporary file for update lists

See Also

sh(1), csh(1), stat(2)

Name

refer – find and format bibliographic references

Syntax

refer [**–a**] [**–b**] [**–c**] [**–e**] [**–f***n*] [**–k***x*] [**–l***m,n*] [**–n**] [**–p** *bib*] [**–s***keys*] [**–B***l.m*] [**–P**] [**–S**] [*file...*]

Description

The `refer` command is a preprocessor for `*roff` that finds and formats references for footnotes or endnotes. It is also the base for a series of programs designed to index, search, sort, and print stand-alone bibliographies, or other data entered in the appropriate form.

Given an incomplete citation with sufficiently precise keywords, `refer` will search a bibliographic database for references containing these keywords anywhere in the title, author, journal, and so forth. The input file (or standard input) is copied to standard output, except for lines between .[and .] delimiters, which are assumed to contain keywords, and are replaced by information from the bibliographic database. The user may also search different databases, override particular fields, or add new fields. The reference data, from whatever source, are assigned to a set of `*roff` strings. Macro packages such as `ms`(7) print the finished reference text from these strings. By default references are flagged by footnote numbers.

Options

The following options are available:

–a*r* Reverses order of first author names. For example, Jones, J. A. instead of J. A. Jones. If *n* is omitted all author names are reversed.

–B*l.m* Bibliography mode. Take a file composed of records separated by blank lines, and turn them into ***roff** input. Label *l* will be turned into the macro *.m* with *l* defaulting to **%X** and *.m* defaulting to **.AP** (annotation paragraph).

–b Creates bare entries: no flags, numbers, or labels.

–c*keys* Capitalizes fields whose key letters are in string.

–e Accumulates all references in one list. Default is to create references where encountered in text. Accumulate them until a sequence of the form

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far.

–f*n* Set the footnote number to *n* instead of the default of 1 (one). With labels rather than numbers, this flag is a no-op.

–k*x* Uses specified label in place of numbering for each reference data line beginning *% x* :. By default *x* is **L.**

–l*m,n*	Instead of numbering references, use labels made from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either *m* or *n* is omitted the entire name or date respectively is used.
–P	Places punctuation marks .,:;?! after the reference signal, rather than before. (Periods and commas used to be done with strings.)
–n	Do not search the default file /usr/dict/papers/Ind. If there is a REFER environment variable, the specified file will be searched instead of the default file; in this case the **–n** flag has no effect.
–p*bib*	Specifies file to be searched before `/usr/dict/papers`.
–S	Produce references in the Natural or Social Science format.
–s*keys*	Uses specified key in sorting references. Implies **–e**. The key-letters in *keys* may be followed by a number to indicate how many such fields are used, with + taken as a very large number. The default is **AD** which sorts on the senior author and then date; to sort, for example, on all authors and then title use **–sA+T**.

To use your own references, put them in the format described below. They can be searched more rapidly by running `indxbib`(1) on them before using `refer`. Failure to index results in a linear search. When `refer` is used with the `eqn`, `neqn` or `tbl` preprocessors `refer` should be first, to minimize the volume of data passed through pipes.

The `refer` preprocessor and associated programs expect input from a file of references composed of records separated by blank lines. A record is a set of lines (fields), each containing one kind of information. Fields start on a line beginning with a "%", followed by a key-letter, then a blank, and finally the contents of the field, and continue until the next line starting with "%". The output ordering and formatting of fields is controlled by the macros specified for `*roff` (for footnotes and endnotes) or `roffbib` (for stand-alone bibliographies). For a list of the most common key-letters and their corresponding fields, see `addbib`(1). An example of a `refer` entry is given below.

Restrictions

Blank spaces at the end of lines in bibliography fields will cause the records to sort and reverse incorrectly. Sorting large numbers of references causes a core dump.

Examples

```
%A   M. E. Lesk
%T   Some Applications of Inverted Indexes on the UNIX System
%B   UNIX Programmer's Manual
%V   2b
%I   Bell Laboratories
%C   Murray Hill, NJ
%D   1978
```

Files

/usr/dict/papers	directory of default publication lists
/usr/lib/refer	directory of companion programs

See Also

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

refile (1mh)

Name

refile – file message in other folders

Syntax

refile [*msgs*] **[–draft] [–link] [–nolink] [–preserve] [–nopreserve] [–src** *+foldername*] **[–file** *filename*] *+folder* **[–help]**

Description

Use the `refile` command to move the specified message from the current folder to another folder or a number of other folders.

If you do not specify a message, the current message will be refiled. Use `refile` in conjunction with the message numbers, to refile a message other than the current message. You can refile more than one message at a time by specifying:

- more than one message number
- a range of message numbers
- a sequence of messages defined by `pick` (see `pick`(1mh))

The following example shows how to refile messages 3 and 5 in the folder +records.

```
$ refile 3 5 +records
```

Options

You will normally `refile` messages from the current folder into another folder. However, you can specify an alternative source folder using the `-src`+folder option. The following example shows how to `refile` a message from the `+inbox` folder when `+inbox` is not the current folder.

```
$ refile 3 –src +inbox +outbox
```

You can `refile` a message in more than one folder, as the following example shows. This means that you could reference mail both under the name of the sender and the subject that the message deals with.

```
$ refile +jones +map
```

Note that folder names are case sensitive, so you must type the exact folder name including any capital letters.

When you `refile` a message, the message is normally `(-nolink)` moved from the source folder to the destination folder. You can keep a copy of the message in the source folder, if you want, by specifying the `-link` option. The following example takes the thirteenth message in the current folder and refiles it in the `+test` folder. Message 13 however remains in the current folder as well as appearing in the `+test` folder.

```
$ refile -link 13 +test
```

Normally when you `refile` a message, the message number is reset to the next available number. So if you were refiling message number 109 into a folder that only contained five messages, the message number would probably not be the same in the new file as it was in the old file. This happens because the `-nopreserve`

option of `refile` is normally in force. You can make sure that the message that you are refiling has the same number in the destination folder as it did in the source folder by specifying the `-preserve` option. You cannot have two messages with the same message number, so you need to use this option carefully.

You can use `refile` to move a file out of a directory into a message folder using the `-file` *filename* option, provided that the file is formatted as a mail message. This means that the message must have the minimum header fields separated from the body of the message by a blank line or a line of dashes. The following example shows this option being used to refile the file <mailfile> into the +test folder.

```
$ refile -file mailfile +test
```

You can `refile` the draft message, or the current message in your drafts file, if you use `refile` with the `-draft` option.

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine your MH directory
Current-Folder:	To find the default current folder
Folder-Protect:	To set mode when creating a new folder
rmmproc:	Program to delete the message

Context

If `-src +folder` is given, it will become the current folder. If neither `-link` nor `all` is specified, the current message in the source folder will be set to the last message specified; otherwise, the current message will not be changed.

If the Previous-Sequence profile entry is set, in addition to defining the named sequences from the source folder, `refile` will also define those sequences for the destination folders. See `mh_profile`(5mh) for information concerning the previous sequence.

See Also

folder(1mh), mh_profile(5mh)

repl(1mh)

Name

repl – reply to a message

Syntax

repl [+*folder*] [*msg*] [**–annotate**] [**–noannotate**] [**–cc** *all/to/cc/me*] [**–nocc** *all/to/cc/me*] [**–draftfolder** +*foldername*] [**–draftmessage** *msg*] [**–nodraftfolder**] [**–editor** *editor*] [**–noedit**] [**–fcc** +*foldername*] [**–filter** *filterfile*] [**–form** *formfile*] [**–format**] [**–noformat**] [**–inplace**] [**–noinplace**] [**–query**] [**–noquery**] [**–width** *columns*] [**–whatnowproc program**] [**–nowhatnowproc**] [**–help**]

Description

You can use the `repl` command to reply to a specified message. If you do not specify a message, `repl` will create a reply to the current message. You can reply to messages other than the current message by specifying the message number. The message replied to will become the current message.

You normally reply to a message in the current folder. However, you can reply to a message in another folder by using the `-folder`<foldername> option. If you do specify a folder, that folder will become the current folder.

When you reply to a message, `repl` automatically fills in the mail header for you, taking the information it needs from the mail header of the original message. The following example shows how `repl` constructs the mail header for the return message.

```
To: <Reply-To> or <From> or <sender>
cc: <cc>, <To>, and yourself
Subject: Re: <Subject>
In-reply-to: Your message of <Date>.
<Message-Id>
```

Field names enclosed in angle brackets (< >) indicate the contents of the named field from the original message.

Options

You can create a different mail header from the default and use it every time you reply to mail. If you create an alternative mail header and store it in the file `replcomps` in your `MH` directory, `repl` will use it instead of the default header file. You can also vary the appearance of the mail header for the reply, by using the `-form`<formfile> option. If you specify a header file with the `form`<formfile> option, this header file will always be used regardless of how the other options are set.

Normally, `repl` will reply to the original sender and send carbon copies to all the addressees on the original message. You can change this, if you want, by specifying the `-cc`<type> option. The `cc` option lets you select the following type switches.

all	replies to all addresses on the original message
to	replies to all addresses in the To: field
cc	replies to all addressees in the cc: field
me	sends you a copy of the reply

In addition to this you can modify the list of recipients by specifying the -query option. This option modifies the action of the -cc type switch by interactively asking you if each address that would normally be placed in the To: and cc: list should actually receive a copy.

You cannot reply to a message if you have a message in your draft file, until you have cleared the draft file. If the draft already exists, repl will ask you for the disposition of the draft. A reply of quit will abort repl, leaving the draft intact; replace will replace the existing draft with a blank skeleton; and list will display the draft.

You can avoid this by setting up a +drafts folder. This will allow you to have more than one incomplete or unsent message in your system at the same time. See folders(1mh) for details of how you can do this.

You can also avoid this problem by specifying -draftfolder *+foldername*. This option lets you determine which folder the message reply is created in. In the following example the message reply would be created in the +answer folder.

```
% repl -draftfolder +answer
```

This reply will be created regardless of whether there is a message already in the draft folder.

Finally, if you do not have a drafts folder set up, you can use the -draftmessage *filename* option. This option allows you to specify a file in which the draft message will be created.

The editor that is provided with repl is prompter. See prompter(1mh) for more details of this editor. You can specify an alternative editor using the -editor<editorname> option. If you are regularly going to use the same editor, you can specify this by putting the following line in your .mh-profile.

```
% editor: <editorname>
```

See mh-profile(5mh) for more details. The -noedit switch can be used to call repl without an editor. See comp(1mh) for more details.

Note that while in the editor, the message being replied to is available through a link named "@" (assuming the default whatnowproc). In addition, the actual pathname of the message is stored in the envariable $editalt, and the pathname of the folder containing the message is stored in the envariable $mhfolder.

Although repl uses the -form *formfile* switch to direct it how to construct the beginning of the draft, the -filter *filterfile* switch tells repl how the message, being replied to, should be formatted in the body of the draft. If -filter is not specified, then the message being replied to is not included in the body of the draft. If -filter filterfile is specified, then the message being replied to is filtered (re-formatted) prior to being output to the body of the draft. The filter file for repl should be a standard form file for mhl, as repl will invoke mhl to format the message being replied to. There is no default message filter. The -filter option must be followed by a file name.

A filter file that is commonly used is:

```
:
body:nocomponent,compwidth=9,offset=9
```

This outputs a blank line and then the body of the message being replied to, indented by one tab-stop.

If the `-annotate` switch is given, the message being replied to will be annotated with the lines

```
Replied: date
Replied: addrs
```

where the address list contains one line for each addressee. The annotation will be done only if the message is sent directly from `repl`. If the message is not sent immediately from `repl`, `comp -use` may be used to re-edit and send the constructed message, but the annotations will not take place. The `-inplace` switch causes annotation to be done in place in order to preserve links to the annotated message.

The `-format` switch specifies that Internet-style formatting should be used. This is the default set up for address formatting. If `-format` is specified, then lines beginning with the fields `To:`, `cc:` and `Bcc:` will be standardized and have duplicate addresses removed.

You can use `-noformat` in conjunction with the `-width` option to format your own address header. However, this can result in creating a message header that will not be accepted by MH or other mail systems. You should only attempt to change the default setting from `-format` to `-noformat` if you completely understand the implications of any changes you intend to make.

The `-width` option governs the maximum width of the header line. Lines exceeding this width are split.

The `-fcc +folder` switch can be used to automatically specify a folder to receive Fcc:s. More than one folder, each preceded by `-fcc` can be named.

Upon exiting from the editor, `repl` will invoke the `whatnow` program. See `whatnow`(1mh) for a discussion of available options. The invocation of this program can be inhibited by using the `-nowhatnowproc` switch. But as it is actually the `whatnow` program which starts the initial edit, specifying `-nowhatnowproc` will prevent any edit from occurring.

If the `whatnowproc` is `whatnow`, then `repl` uses its own built-in `whatnow`, it does not actually run the `whatnow` program. Hence, if you define your own `whatnowproc`, do not call it `whatnow` since `repl` will not run it.

If your current working directory is not writable, the link named `@` is not available. The default settings for this command are:

+folder defaults to the current folder
msg defaults to the current message
`-cc` all
`-format`
`-noannotate`
`-nodraftfolder`
`-noinplace`
`-noquery`
`-width` 72

Files

/usr/new/lib/mh/replcomps	The reply template
<mh-dir>/replcomps	Alternative to the standard reply template
$HOME/.mh_profile	The user profile
<mh-dir>/draft	The draft file

Profile Components

Path:	To determine your MH directory
Alternate-Mailboxes:	To determine your mailboxes
Current-Folder:	To find the default current folder
Draft-Folder:	To find the default draft–folder
Editor:	To override the default editor
Msg-Protect:	To set mode when creating a new message (draft)
fileproc:	Program to refile the message
mhlproc:	Program to filter message being replied to
whatnowproc:	Program to ask the "What now?" questions

See Also

comp(1mh), dist(1mh), forw(1mh), prompter(1mh), send(1mh), whatnow(1mh), mh-format(5mh)

reset(1)

Name

reset – reset terminal mode

Syntax

reset

Description

The `reset` command sets the terminal to cooked mode, turns off cbreak and raw modes, turns on nl, and restores special characters that are undefined to their default values. The `reset` command also clears the LLITOUT bit in the local mode word.

This is most useful after a program dies leaving a terminal in a funny state; you have to type "<LF>reset<LF>" to get it to work then to the shell, as <CR> often doesn't work; often none of this will echo.

It is a good idea to follow `reset` with `tset`(1)

Restrictions

Doesn't set tabs properly. It can't interpret choices for interrupt and line kill characters, so it leaves these set to the local system standards.

See Also

stty(1), tset(1)

Name

rev – reverse character positions in file data

Syntax

rev [*file...*]

Description

The `rev` command copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

rlogin (1c)

Name

rlogin – remote login

Syntax

rlogin *rhost* [**–e** *c*] [–8] [**–L**] [**–l** *username*]
rhost [**–e** *c*] [–8] [**–L**] [**–l** *username*]

Description

The `rlogin` command connects your terminal on the current local host system, *lhost,* to the remote host system, *rhost.*

Each host has a file `/etc/hosts.equiv` which contains a list of *rhosts* with which it shares account names. The host names must be the standard names as described in `rsh`(1c). When you use the `rlogin` command to login as the same user on an equivalent host, you do not need to specify a password.

You can also have a private equivalence list in a file .rhosts in your login directory. Each line in this file should contain the *rhost* name and a *username* separated by a space, giving additional cases where logins without passwords are permitted. If the originating user is not equivalent to the remote user, then the remote system prompts for a login and password as in `login`(1).

To avoid security problems, the .rhosts file must be owned by either the remote user or root and it may not be a symbolic link.

Your remote terminal type is the same as your local terminal type, which is specified by your environment TERM variable. Except for delays, all echoing takes place at the remote site so the rlogin is transparent. Flow control by <CTRL/S> and <CTRL/Q>, and flushing of input and output on interrupts are handled properly. The optional argument –8 allows an eight-bit input data path at all times. Otherwise, parity bits are stripped except when the remote site's stop and start characters are other than <CTRL/S> and <CTRL/Q>. A tilde followed by a dot (~.) on a separate line disconnects from the remote host, where the tilde (~) is the escape character. Similarly, a tilde followed by <CTRL/Z> (~ <CTRL/Z>), where <CTRL/Z> is the suspend character, suspends the rlogin session.

Substitution of the delayed-suspend character, which is normally <CTRL/Y>, for the suspend character suspends the send portion of the rlogin, but allows output from the remote system. A different escape character may be specified by the **–e** option. There is no space separating this option flag and the argument character.

Options

–8	Allows an 8-bit input data path at all times.
–e*c*	Uses the specified character as the `rlogin` escape character. If not specified, uses a tilde (~).
–l *username*	Logs you in as the specified user, not as your user login name.
–L	Runs session in litout mode.

Files

/usr/hosts/*	for *rhost* version of the command

See Also

rsh(1c)

rm(1)

Name

rm, rmdir – remove (unlink) files or directories

Syntax

rm [**–f**] [**–r**] [**–i**] [**–**] *file-or-directory-name...*
rmdir *directory-name...*

Description

The `rm` command removes the entries for one or more files from a directory. If there are no links to the file then the file is destroyed. For further information, see `ln`(1).

The `rmdir` command removes entries for the named directories, which must be empty. If they are not empty, the directories remain, and `rmdir` displays an error message (see EXAMPLES).

To remove a file, you must have write permission in its directory, but you do not need read or write permission on the file itself. When you are using `rm` from a terminal, and you do not have write permission on the file, the `rm` command asks for confirmation before destroying the file.

If input is redirected from the standard input device (your terminal), then `rm` checks to ensure that input is not coming from your terminal. If not, `rm` sets the **–f** option, which overrides the file protection, and removes the files silently, regardless of what you have specified in the file redirected as input to `rm`. See EXAMPLES.

Options

– Specifies that the named files have names beginning with a minus (for example `-myfile`).

–f Forces the removal of file or directory without first requesting confirmation. Only system or usage messages are displayed.

–i Prompts for yes or no response before removing each entry. Does not ask when combined with the **–f** option. If you type a y, followed by any combination of characters, a yes response is assumed.

–r Recursively removes all entries from the specified directory and, then, removes the entry for that directory from its parent directory.

Examples

The following example shows how to remove a file in your current working directory.

```
rm myfile
```

This example shows use of the null option to remove a file beginning with a minus sign.

```
rm - -gorp
```

This example shows how a confirmation is requested for removal of a file for which you do not have write permission.

```
rm testfile
```

```
rm: override protection 400 for testfile? y
```

This example shows how the combination of **–i** and **–r** options lets you examine all the files in a directory before removing them. In the example, mydirectory is a subdirectory of the current working directory. Note that the last question requests confirmation before removing the directory itself. Although the user types "y", requesting removal of the directory, the `rm` command does not allow this, because the directory is not empty; the user typed "n" to the question about the file *file2* , so *file2* was not removed.

```
rm -ir mydirectory
rm: remove mydirectory/file1? y
rm: remove mydirectory/file2? n
      .
      .
      .
rm: remove mydirectory? y
rm: mydirectory: Directory not empty
```

This example illustrates that `rm` overrides file protection when input is redirected from the standard input device. The user creates a file named "alfie", with a read-only file protection. The user then creates a file named "ans" to contain the character "n". The `rm` command following destroys the file "alfie", even though the redirected input file requested no deletion.

```
cat > alfie
hello
^d
chmod 444 alfie
cat > ans
n
^d
rm < ans alfie
```

See Also

unlink(2)

rmail(1)

Name

rmail – route mail to users on remote systems

Syntax

rmail *user...*

Description

The `rmail` command interprets incoming mail received via `uucp`(1c), collapsing "From" lines in the form generated by `binmail`(1) into a single line of the form "return-path!sender", and passing the processed mail on to `sendmail`(8).

The `rmail` is explicitly designed for use with `uucp` and `sendmail`.

See Also

binmail(1), uucp(1c), sendmail(8)

Name

rmdel – remove a delta from an SCCS file

Syntax

rmdel –r *files*

Description

The `rmdel` command removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the most recent delta in its branch in the delta chain of each named SCCS file. In addition, the SID specified delta must not be that of a version being edited for the purpose of making a delta, for example, if a *p-file* exists for the named SCCS file, the SID specified delta must not appear in any entry of the *p-file*). For further information, see `get`(1).

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 – is given as the name, the standard input is read and 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.

Certain permissions are necessary to remove a delta. If you make a delta or own the file and directory, you can remove it.

Options

–r*SID* Specifies the delta version number.

Diagnostics

Use `sccshelp`(1) for explanations.

Files

x-file For further information, see `delta`(1).
z-file For further information see `delta`(1).

See Also

delta(1), get(1), help(1), prs(1), sccs(1), sccsfile(5)
Guide to the Source Code Control System

rmf(1mh)

Name

rmf – remove folder

Syntax

rmf [*+foldername*] **[–interactive] [–nointeractive] [–help]**

Description

The `rmf` command removes all of the messages (files) within the current folder and then removes the folder itself. If there are any files within the folder which are not part of `MH`, they are not removed, and an error message is displayed.

You can specify a folder other than the current folder, by using the `+folder` option. If you do not specify a folder, and `rmf` cannot find the current folder, `rmf` asks you whether you want to delete `+inbox` instead.

Note that the `rmf` command irreversibly deletes messages that do not have other links, so use it with caution.

If the folder being removed is a subfolder, the parent folder becomes the new current folder, and `rmf` tells you that this has happened. This provides an easy mechanism for selecting a set of messages, operating on the list, then removing the list and returning to the current folder from which the list was extracted.

The `rmf` of a read-only folder deletes the private sequence and `cur` information (`atr-`, `seq-` and `folder` entries) from the profile without affecting the folder itself. If you have sub-folders within a folder, you must delete all the sub folders before you can delete the folder itself.

Files

$HOME/.mh_profile The user profile

Profile Components

Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder

Context

This command sets the current folder to the parent folder if a sub-folder is removed. If the current folder is removed, it makes `inbox` current. In all other cases it does not change the current folder or message.

See Also

rmm(1mh)

Name

rmm – remove messages

Syntax

rmm [*+folder*] [*msgs*] **[–help]**

Description

The `rmm` command deletes the current message from the current folder.

It removes messages by renaming the message files with preceding commas. Many sites consider files that start with a comma to be a temporary backup, and arrange for `cron`(8) to remove such files once a day.

You can specify messages and folders other than the current ones by using the `+folder` and `msgs` arguments.

If you have a profile component such as

```
rmmproc:      /bin/rm
```

then instead of simply renaming the message file, `rmm` will call the named program to delete the file.

The current message is not changed by `rmm`, so a `next` will advance to the next message in the folder as expected.

The default settings for this command are:
+folder defaults to the current folder
msgs defaults to the current message

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine your MH directory
Current–Folder:	To find the default current folder
rmmproc:	Program to delete the message

See Also

rmf(1mh)

roffbib(1)

Name

roffbib – run off bibliographic database

Syntax

roffbib [*options*] [*file...*]

Description

The `roffbib` command prints out all records in a bibliographic database, in bibliography format rather than as footnotes or endnotes. Generally it is used in conjunction with `sortbib`:

```
sortbib  database | roffbib
```

Options

The `roffbib` command accepts most of the options understood by nroff(1).

–T*term* Uses specified name as terminal type for which output is prepared.

–x Suppresses the printing of abstracts.

If abstracts or comments are entered following the %X field key, `roffbib` formats them into paragraphs for an annotated bibliography. Several %X fields may be given if several annotation paragraphs are desired.

–e Formats text with equally spaced words, justified lines, and full resolution.

–h Uses tabs in horizontal spacing to speed output and reduce output character count. Tab characters are assumed to be every 8 nominal character widths.

–n Uses specified number (**–n***N*) as first page to be printed.

–o Uses specified list (**–o***list*) as only pages to be printed. A range *N–M* means pages *N* through *M*. An initial *–N* means from the beginning to page *N*. A final *N–* means from *N* to the end.

–s Stops after specified number of pages (**–s***n*).

–m *mac* Specifies a user-defined set of macros with space between **–m** and the macro file name. This set of macros replaces the ones defined in /usr/lib/tmac/tmac.bib.

–V Sends output to the Versatec.

–Q Queues output for the phototypesetter.

–r*aN* Sets named register *a* to specified value *N*.

Four command-line registers control formatting style of the bibliography, much like the number registers of ms(7). The command-line argument **–r**N1 numbers the references starting at one (1). The flag **–r**V2 double spaces the bibliography, while **–r**V1 double spaces references but single spaces annotation paragraphs. The line length can be changed from the default 6.5 inches to 6 inches with the **–r**L6i argument, and the page offset can be set from the default of 0 to one inch by specifying **–r**O1i (capital O, not zero). Note: with the **–V** and **–Q** flags the default page offset is already one inch.

Files

/usr/lib/tmac/tmac.bib file of macros used by *nroff/troff*

See Also

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

rsh(1c)

Name

rsh – remote shell

Syntax

rsh *host* [**–l** *username*] [**–n**] *command*
host [**–l** *username*] [**–n**] *command*

Description

The `rsh` command connects to the specified *host,* and executes the specified *command.* The `rsh` command copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command. The `rsh` command normally terminates when the remote command does.

The remote username used is the same as your local username, unless you specify a different remote name with the **–l** option. This remote name must be equivalent, in the sense of `rlogin`(1c), to the originating account. No provision is made for specifying a password with a command.

If you omit *command,* then instead of executing a single command, you are logged in on the remote host using `rlogin`(1c).

Shell metacharacters which are not quoted are interpreted on local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command

```
rsh otherhost cat remotefile >> localfile
```

appends the remote file *remotefile* to the localfile *localfile,* while

```
rsh otherhost cat remotefile ">>" otherremotefile
```

appends *remotefile* to *otherremotefile.*

Host names are given in the file `/etc/hosts`. Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames. The host names for local machines are also commands in the directory `/usr/hosts`. If you put this directory in your search path then the `rsh` can be omitted.

Options

–l *username*	Logs you in as the specified user, not as your user login name.
–n	Redirects all command input to `/dev/null`.

Restrictions

If you are using `csh`(1) and put a `rsh`(1c) in the background without redirecting its input away from the terminal, it blocks even if no reads are posted by the remote command. If no input is desired you should redirect the input of `rsh` to `/dev/null` using the **–n** option.

You cannot run an interactive command like `vi`(1). Use `rlogin`(1c).

Stop signals stop the local `rsh` process only.

Files

/etc/hosts
/usr/hosts/*

See Also

rlogin(1c)

ruptime(1c)

Name

ruptime – show host status of local machines

Syntax

ruptime [*options*] [*machinename*]

Description

The `ruptime` command gives a status line like `uptime`(1) for each machine on the local network. If a *machinename* is given, the status of only the named machine is given. These status lines are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 5 minutes are shown as being down.

Options

–a Users idle an hour or more are not counted unless this option is specified.

–d Display only those hosts that are considered down.

–l Sort the status list by load average. If more than one sort option is given, `ruptime` uses the last one.

–r Show only hosts that are up and running.

–t Sort the status list by uptime. If more than one sort option is given, `ruptime` uses the last one.

–u Sort the status list by number of users. If more than one sort option is given, `ruptime` uses the last one.

–nn Show only those hosts with *nn* or more users.

Restrictions

Because the `rwhod` daemon sends its information in broadcast packets it generates a large amount of network traffic. On large networks the extra traffic may be objectionable. Therefore, the `rwhod` daemon is disabled by default. To make use of the `rwhod` daemon for both the local and remote hosts, remove the comment symbols (#) from in front of the lines specifying `rwhod` in the `/etc/rc` file.

If the `rwhod` daemon is not running on a remote machine, the machine may incorrectly appear to be down when you use the `ruptime` command to determine its status. See the `rwhod`(8) reference page for more information.

If a system has more than 40 users logged in at once, the number of users displayed by the `ruptime` command is incorrect. Users who login to a machine after that point fail to increment the user count that appears in the output of the `ruptime` command. This is due to the maximum size limit of an Ethernet packet, which is 1500 bytes, and the fact that the `rwhod` daemon must broadcast its information in a single packet.

Files

/usr/spool/rwho/whod.* Information about other machines

See Also

rwho(1c), rwhod(8c)

rwho(1c)

Name

rwho – who is logged in on local machines

Syntax

rwho [**–ah**] [*users*]

Description

The `rwho` command lists the login name, terminal name, and login time for users on all machines on the local network. If no report has been received from a machine for 5 minutes, `rwho` assumes that the machine is down, and does not report users last known to be logged in to that machine. If a user has not typed to the system for a minute or more, `rwho` reports this idle time.

If a user has not typed to the system for an hour or more, the user is omitted from the output of `rwho`.

If given a list of user names, the `rwho` command reports on the status of only those names.

Options

–a Lists all users. Normally, `rwho` omits users who have not typed to the system for an hour or more. If the **–a** flag is specified, these users are also listed.

–h Sorts users by host name. Normally, `rwho` prints its output sorted by user name. If the **–h** flag is specified, the results are sorted by host name.

Files

/usr/spool/rwho/whod.* Information about other machines

See Also

ruptime(1c), rwhod(8c)

Name

sact – display current SCCS file editing activity

Syntax

sact *files*

Description

The `sact` command informs the user of any impending deltas to a named SCCS file. This situation occurs when the `get` command with the **–e** option has been previously executed without a subsequent execution of the `delta` command. 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 login name 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.

Diagnostics

See `sccshelp`(1) for explanations.

See Also

delta(1), get(1), help(1), sccs(1), sccsfile(5), sccshelp(1), unget(1)
Guide to the Source Code Control System

scan(1mh)

Name

scan – produce a one-line-per-message scan listing

Syntax

scan [*+folder*] [*msgs*] [**–clear**] [**–noclear**] [**–form** *formatfile*] [**–format** *string*] [**–header**] [**–noheader**] [**–width** *columns*] [**–help**]

Description

The `scan` command produces a one-line-per-message listing of the messages in the current folder. You can `scan` a folder other than the current folder by using the *+foldername* argument.

You can use the *msgs* argument to produce a scan listing of a number of messages or a range of messages in the specified folder. The following example would produce a listing of messages 10 through 20 in the folder +inbox.

```
$ scan +inbox 10–20
```

You can also use `scan` in conjunction with a message sequence defined by `pick`. See `pick`(1mh) for details.

Each `scan` line contains the message number (name), the date, the `From:` field, the `Subject:` field, and, if room allows, some of the body of the message. For example:

```
15+    7/ 5   Dcrocker      Volunteers <<Last week I asked
16 -   7/ 5   dcrocker      message id format <<I recommend
18     7/ 6   Obrien         Re: Exit status from mkdir
19     7/ 7   Obrien         ``scan'' listing format in MH
```

The + on message 15 indicates that it is the current message. The - on message 16 indicates that it has been replied to, as indicated by a `Replied:` component produced by an `-annotate` switch to the `repl` command. If there is sufficient room left on the `scan` line after the subject, the line will be filled with text from the body, preceded by <<, and terminated by >> if the body is sufficiently short.

`Scan` actually reads each of the specified messages and parses them to extract the desired fields. During parsing, appropriate error messages will be produced if there are format errors in any of the messages.

Options

The `-header` switch produces a header line prior to the `scan` listing. The header line displays the name of the folder and the date and time.

If the -clear switch is used and scan's output is directed to a terminal, then scan will consult the $TERM and $TERMCAP envariables to determine your terminal type in order to find out how to clear the screen prior to exiting. If the -clear switch is used and scan's output is not directed to a terminal, then scan will send a formfeed prior to exiting.

In the following example, the scan command produces a scan listing of the current folder, followed by a formfeed, followed by a formatted listing of all messages in the folder, one per page. Omitting -show pr -f will cause the messages to be concatenated, separated by a one-line header and two blank lines.

```
(scan -clear -header; show all -show pr -f) | lpr
```

If scan encounters a message without a Date: field, the date is filled in with the last write date of the message, and post-fixed with a *. This is particularly useful for scanning a draft folder, as message drafts usually are not allowed to have dates in them.

To override the output format used by scan, the -format string or -format file switches are used. This permits individual fields of the scan listing to be extracted with ease. The string is simply a format string and the file is simply a format file. See mh-format(5mh) for more details.

Because MH has been configured with the BERK option, scan has two other switches: -reverse and -noreverse. These make scan list the messages in reverse order. In addition, scan will update the MH context prior to starting the listing, so interrupting a long scan listing preserves the new context. The default configuration file that is supplied with MH has the BERK option enabled.

The defaults for this command are:

+*folder* defaults to the folder current
msgs defaults to all
-format defaulted as described above
-noheader
-width defaulted to the width of the terminal

The argument to the -format switch must be interpreted as a single token by the shell that invokes scan. Therefore, you should place the argument to this switch inside double quotes (" ").

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine your MH directory
Alternate-Mailboxes:	To determine your mailboxes
Current-Folder:	To find the default current folder

See Also

inc(1), pick(1), show(1), mh-format(5)

scat (1)

Name

scat – sparse data file utility

Syntax

scat –c *file1 ... filen*
scat –d *file1 ... filen*
scat [–m] *file1 file2*

Description

The `scat` command copies, moves, compresses, or decompresses sparse data files.

Options

–c Compresses or reduces the size of sparse data files and makes them contiguous data files. Use the **–c** option, for example, when you want to put sparse data files on external media such as magnetic tape. Each sparse data file is replaced by one with the extension `.S`, while keeping the same ownership modes, access, and modification times. If no files are specified, the standard input is compressed to the standard output. Compressed files can be restored to their original form using the **–d** option.

–d Decompresses or expands compressed sparse data files and makes them sparse again. Use the **–d** option to restore compressed sparse data files to sparse data files. This option looks for files with the extension `.S`, and if the files are valid compressed sparse data files and then decompresses them. The decompressed files will have the `.S` extension removed.

–m Moves or renames a sparse data file.

See Also

savecore(8)

Name

sccs – Source Code Control System

Syntax

sccs [*flags*] *command* [*command-option*] [*file*] [sccs]

Description

SCCS is a source management system which maintains records of changes made in files within that system. Records stating what the changes were, why and when they were made, and who made them are kept for each version. Previous versions can be recovered, and different versions can be maintained simultaneously. SCCS also insures that two people are not editing the same file at the same time.

The SCCS system has two levels of operation, a preprocessor called `sccs` and the traditional SCCS commands. The preprocessor `sccs`(1) provides an interface with the more traditional SCCS commands, such as **get, delta,** and so forth. The `sccs`(1) interface is a more user-friendly environment for the SCCS user. Some of the commands are more intuitive, such as **sccs edit** rather than the traditional **get -e** to retrieve a file for editing. Some commands perform multiple operations, such as **sccs delget** which performs a **delta** on the file, and then **get** the changed file back.

The `sccs`(1) preprocessor also restructures the method in which SCCS files are stored and manipulated. In the traditional version of SCCS, files (s-files, p-files, and so forth) are stored in the directory that contains the g-files, unless an SCCS directory is explicitly defined with each command. The `sccs`(1) preprocessor expects that an SCCS directory is available within the directory that contains the g-files and that this directory also contains the SCCS files. The SCCS directory is owned by `sccs`, providing an additional level of security. This method also cleans up the directory where the g-files are stored. One other important difference in using the preprocessor is that the file specification is the name of the g-file rather than the name of the s-file when invoking an SCCS command. If SCCS is specified rather than the file name, the SCCS preprocessor handles this in the same manner as the traditional commands handle it. Each s-file in the directory SCCS is acted upon as if explicitly named. Please note that not all `sccs`(1) preprocessor commands permit this feature.

The traditional SCCS commands are also included for reference in SEE ALSO. If the `sccs`(1) preprocessor is used, small discrepancies may exist due to conflicts between the command parameters and the `sccs`(1) preprocessor parameters.

SCCS stores all versions of each file along with the logged data about each version in the s-file. Three major operations can be performed on the s-file.

To retrieve a file for reading or printing use the following command:

```
sccs get [filename][sccs]
```

The latest version is retrieved and is NOT intended for edit.

To retrieve a file for edit use the following command:

```
sccs edit [filename][sccs]
```

The latest version is retrieved and only one person can edit a given file at one time.

To merge a file back into the s-file use the following command:

```
sccs delta [filename][sccs]
```

This is a companion operation to the edit command (Step 2). A new version number is assigned. Comments explaining the changes are saved.

A **delta** consists of the changes made to a file, not the file itself. All deltas to a given file are stored, enabling you to get a version of the file that has selected deltas removed which gives you the option of removing your selected changes later.

An SID is an identification number for a delta. It consists of two parts, a release number and a level number. The release number normally remains constant but can be changed when major changes in the file are made. The level numbers represent each delta for a given file. A SID can also be used to represent a version number of the entire file.

To create all the source files in a given directory in SCCS format, run the following shell script from csh:

```
mkdir SCCS save
chown sccs SCCS
foreach i (*.[.ch])
        sccs admin -i$i $i
        mv $i save/$i
end
```

Note that to run the `chown` command, you must be the superuser. However, the *Guide to the Source Code Control System* describes a method for setting up SCCS files that makes superuser privileges unnecessary.

To create a single source file in SCCS format, assuming the presence of the SCCS directory, use the following command:

```
sccs create <filename>
```

Because the number and types of commands used within SCCS are many and complex, the following quick reference table is included here. See the Guide to the Source Code Control System for further explanations of commands.

Flags

–d<dir> The <dir> represents a directory to search out of. It should be a full pathname for general usage. For example, if <dir> is `/usr/src/sys`, then a reference to the file `dev/bio.c` becomes a reference to `/usr/src/sys/dev/bio.c`.

–p<path> Prepends <path> to the final component of the pathname. By default, this is SCCS. For example, in the **–d** example above, the path then gets modified to `/usr/src/sys/dev/SCCS/s.bio.c`. In more common usage (without the –d flag), `prog.c` would get modified to `SCCS/s.prog.c`. In both cases, the `s.` gets automatically prepended.

–r Run as the real user.

Commands

These commands should all be preceded by `sccs`.

get — Gets files for compilation (not for editing). Id keywords are expanded.

- **–e** Gets a writable copy of the file.
- **–r***SID* Get specified version.
- **–p** Send to standard output rather than to the actual file.
- **–k** Gets a writable copy of the file. Does not expand id keywords.
- **–i***list* Include list of deltas.
- **–x***list* Exclude list of deltas.
- **–m** Precede each line with SID of delta being created.
- **–c***date* Do not apply any deltas created after *date*.

edit — Gets files for editing. Id keywords are not expanded. Should be matched with a **delta** command after editing.

- **–r***SID* Get specified version. If SID specifies a release that does not yet exist, the highest numbered delta is retrieved and the new delta is numbered with SID
- **–b** Create a branch.
- **–i***list* Include list of deltas
- **–x***list* Exclude list of deltas

delta — Merge a file retrieved using **edit** back into the s-file. Collect comments about why this delta was made.

unedit — Remove a file that has been edited previously without merging the changes into the s-file.

info — Display a list of all files being edited.

- **–b** Ignore branches.
- **–u**[*user*] Ignore files not being edited by *user* .

check — Same as *info,* except that nothing is printed if nothing is being edited and exit status is returned.

prs — Produces a report of changes to the named file. Time, date, user, number of lines changed, the revision number, and comments are listed for each delta.

create — Create an s. file and do not remove the associated g-file.

tell — Same as *info,* except that only the file name of files being edited is listed.

clean — Remove all files that can be regenerated from the s-file.

what — Find and print id keywords.

admin — Create or set parameters on s-files.

–i*file*	Create, using *file* as the initial contents.
–z	Rebuild the checksum in case the file has been corrupted.
–f*flag*	Turn on the *flag*.
–d*flag*	Turn off (delete) the *flag* .
–t*file*	Replace the text in the s-file with the contents of *file*. If *file* is omitted, the text is deleted. Useful for storing documentation or design and implementation documents to insure distribution with the s-file.

Useful flags are:

b	Allow branches to be made using the -b flag to *edit*.
d*SID*	Default SID to be used on a **get** or **edit.**
i	Cause *No Id Keywords* error message to be a fatal error rather than a warning.
t	The module type; the value of this flag replaces the **%Y%** keyword.

fix	Remove a delta and reedit it.
delget	Do a **delta** followed by a **get.**
deledit	Do a **delta** followed by an **edit.**
diffs	Compare the g-file out for edit with an earlier SCCS version.
sccsdiff	Compare any two SCCS versions of a g-file.
help	Given either a command name, or an sccs message number, this command provides additional information.

Id Keywords

%Z%	Expands to @(#) for the *what* command to find.
%M%	The current module name, for example, prog.c.
%I%	The highest SID applied.
%W%	A shorthand for "%Z%%M% <tab> %I%".
%G%.	The date of the delta corresponding to the %I%. keyword.
%R%.	The current release number, for example, the first component of the %I% keyword.
%Y%	Replaced by the value of the **t** flag (set by **admin).**

See Also

admin(1), cdc(1), comb(1), delta(1), get(1), prs(1), rmdel(1), sccshelp(1), unget(1), val(1), what(1), sccsfile(5)
Guide to the Source Code Control System

Name

sccsdiff – compare and display SCCS delta differences

Syntax

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. You can specify any number of SCCS files, but arguments apply to all files.

Options

–p Displays output using `pr`(1) command.

–r*SID?* Indicates deltas to be prepared. Versions are passed to `bdiff`(1) in the order given.

–s*n* Sets number of lines each segment is to contain. This is useful when `diff` fails due to a high system load.

Diagnostics

"*file***: No differences"**
If the two versions are the same. Use `sccshelp`(1) for explanations.

Files

`/tmp/get?????`
temporary files

See Also

bdiff(1), cmp(1), comm(1), diff(1), diff3(1), diffmk(1), get(1), prs(1), sccs(1), sccshelp(1)
Guide to the Source Code Control System

sccshelp(1)

Name

sccshelp – display SCCS help information

Syntax

sccs help [*args*]
sccshelp [*args*]

Description

The `sccshelp` command explains the use of a sccs command or provides additional information on sccs-generated messages. Zero or more arguments may be supplied to help. If no arguments are given, `sccshelp` will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages), command names, or one of the following types:

type 1
: Begins with nonnumerics, ends in numerics. The nonnumeric prefix is usually an abbreviation for the program or set of routines which produced the message (For example, **ge6**, for message 6 from the **get** command).

type 2
: Does not contain numerics (For example, **get**, a command name)

type 3
: Is all numeric (For example, 212)

When all else fails, try

```
sccs help stuck.
```

Files

/usr/lib/sccs.help
: Directory containing files of message text

/usr/lib/sccs.help/helploc
: File containing locations of help files not in `/usr/lib/help`.

Name

script – generate script of your terminal session

Syntax

script [**–a**] [*file*]

Description

The `script` command calls a forked shell and then makes a typescript of everything printed on your terminal. The typescript is written to *file,* or appended to *file* if the **–a** option is given. If no file name is given, the typescript is saved in the file *typescript* . Later, it can be sent to the line printer with `lpr`.

When `script` is run, it will execute your .cshrc file and will also put you under control of a new pty. The `script` program ends when the forked shell exits.

This program is useful when using a crt and you want a hard-copy record of the dialog. An example of how `script` can be used is a student handing in a program that was developed on a crt when hard-copy terminals are in short supply.

Options

–a Appends output to the output *file.*

Restrictions

Because the `script` command uses a second pair of pseudo ttys, some commands will fail when run under `script`. For example, the `whoami` command may not always return a username.

Also, the `vi` editor can be affected by `script`. The `vi` commands j, k, h, and l that are used to move the cursor, do not always function properly. If the screen becomes garbled when using these commands, use <ctrl> L to refresh the screen.

See Also

pty(4)

sed (1)

Name

sed – stream text editor

Syntax

sed [**–n**] [**–e** *script*] [**–f** *sfile*] [*file...*]

Description

The `sed` command copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **–f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **–e** option and no **–f**'s, the flag **–e** may be omitted. The **–n** option suppresses the default output.

A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation `sed` cyclically copies a line of input into a *pattern space* (unless there is something left after a 'D' command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **–n**) and deletes the pattern space.

An *address* is either a decimal number that counts input lines cumulatively across files, a '$' that addresses the last line of input, or a context address, '/regular expression/', in the style of ed(1) modified thus:

- 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 command line with no addresses selects every pattern space.
- A command line with one address selects each pattern space that matches the address.
- A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function '!' (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

An argument denoted *text* consists of one or more lines, all but the last of which end with '\' to hide the 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.

An argument denoted *rfile* or *wfile* must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) a\
text

Append. Place *text* on the output before reading the next input line.

(2) b *label*

Branch to the ':' command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) c\
text

Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) d Delete the pattern space. Start the next cycle.

(2) D Delete the initial segment of the pattern space through the first 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) 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 more complete description see ed(1). The *flags* is zero or more of

g Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Print the pattern space if a replacement was made.

w *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) t *label*
: Test. Branch to the ':' command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a 't'. If *label* is empty, branch to the end of the script.

(2) w *wfile*
: Write. Append the pattern space to *wfile*.

(2) x
: Exchange the contents of the pattern and hold spaces.

(2) y/*string1*/*string2*/
: Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)! *function*
: Don't. Apply the *function* (or group, if *function* is '{') only to lines *not* selected by the address(es).

(0) : *label*
: This command does nothing; it bears a *label* for 'b' and 't' commands to branch to.

(1) =
: Place the current line number on the standard output as a line.

(2) {
: Execute the following commands through a matching '}' only when the pattern space is selected.

(0)
: An empty command is ignored.

Options

–e *script*
: Uses specified file as input file of commands to be executed.

–f *sfile*
: Uses specified file as input file of commands to be executed. May be used with **–e** option to indicate two script files.

–n
: Suppresses all normal output.

See Also

awk(1), ed(1), grep(1), lex(1)

Name

send – send a message

Syntax

send [–alias *aliasfile*] **[–draft] [–draftfolder** *+foldername*] **[–draftmessage** *msg*] **[–nodraftfolder] [–filter** *filterfile*] **[–nofilter] [–format] [–noformat] [–forward] [–noforward] [–msgid] [–nomsgid] [–push] [–nopush] [–verbose] [–noverbose] [–watch] [–nowatch] [–width** *columns*] **[file ...] [–help]**

Description

Use `send` to send the draft message to the specified recipients. You normally choose `send` as one of the options from the `whatnow` program. However, you can use `send` just like any other `MH` command.

Normally messages are created in the file `draft`. This file is stored in your `Mail` directory. The draft message remains in the file `draft` until it is either sent or deleted.

The command `send` will normally search for the `draft` file and cause it to be delivered to each of the destinations in the `To:`, `cc:`, `Bcc:`, and `Fcc:` fields of the message. If `send` is redistributing a message, as invoked from `dist`, the corresponding `Resent-xxx` fields are examined instead. The delivery is carried out using `post`(8mh).

Options

You can specify alternative messages or files to be used by `send`, provided that they are formatted as legal mail messages. If you use the `-draftfolder` *foldername* option, `send` will search the specified folder for the draft message and will deliver it to the specified recipients. If there is no draft message in the specified folder, `send` will display an error message. In the following example `send` will search the folder `+test` for the draft message and send it if one exists.

```
% send –draftfolder +test
```

If you specify the `-draftmessage` *filename* option, you can `send` a file instead of a folder. If you do not specify a directory, the file will be assumed to be in your `Mail` directory. Note that, if you specify a relative pathname (one that doesn't start with a `/`), `send` will assume that the pathname is relative to your Mail directory. In the following example `send` will search for the file `draftmessage` in user Robb's test directory.

```
% send –draftmessage ~robb/test/draftmessage
```

You should not attempt to use both the `-draftfolder` and the `-draftmessage` options at the same time.

Issuing `send` with no `file` argument will query whether the draft is the intended file, whereas `-draft` will suppress this question. Once the transport system has successfully accepted custody of the message, the file will be renamed with a leading comma. This allows it to be retrieved until the next draft message is sent. If there are errors in the formatting of the message, `send` will abort and issue an error

message.

If a Bcc: field is encountered, its addresses will be used for delivery, and the Bcc: field will be removed from the message sent to sighted recipients. The blind recipients will receive an entirely new message with a minimal set of headers. Included in the body of the message will be a copy of the message sent to the sighted recipients. If you specify -filter *filterfile* then this copy is filtered (re-formatted) prior to being sent to the blind recipients.

Prior to sending the message, the fields From: user@local, and Date: now will be appended to the headers in the message. If the environment variable $SIGNATURE is set, then its value is used as your personal name when constructing the From: line of the message. If this variable is not set, then send will consult the profile entry Signature: for this information.

If -msgid is specified, then a Message-ID: field will also be added to the message.

If send is redistributing a message (when invoked by dist), then "Resent–" will be prepended to each of these fields: "From:", "Date:", and "Message–ID:". If the message already contains a "From:" field, then a "Sender: user@local" field will be added as well.

By using the -format switch, each of the entries in the To: and cc: fields will be replaced with standard format entries. This standard format is designed to be usable by all of the message handlers on the various systems around the Internet. If -noformat is given, then headers are output exactly as they appear in the message draft.

If an Fcc: folder is encountered, the message will be copied to the specified folder for the sender in the format in which it will appear to any normal recipients of the message. That is, it will have the appended fields and field reformatting. The Fcc: fields will be removed from all outgoing copies of the message.

By using the -width *columns* switch, you can specify how long send should make header lines containing addresses.

By using the -alias aliasfile switch, you can direct send to consult the named files for alias definitions (more than one file, each preceded by -alias, can be named). See mh-alias(5mh) for more information.

If -push is specified, send performs its actions in the background. If you specify -push, and -forward and the draft cannot be sent, then the -forward switch says that draft should be forwarded with the failure notice sent to the sender. This differs from putting send in the background because the output is trapped and analyzed by MH.

If -verbose is specified, send will indicate the interactions occurring with the transport system, prior to actual delivery. If -watch is specified send will monitor the delivery of local and network mail. Hence, by specifying both switches, a large amount of information can be gathered about each step of the message's entry into the transport system. The defaults for the send command are:

```
file defaults to <mh–dir>/draft
-alias /usr/new/lib/mh/MailAliases
-nodraftfolder
-nofilter
-format
```

```
-forward
-nomsgid
-nopush
-noverbose
-nowatch
-width 72
```

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine the user's MH directory
Draft-Folder:	To find the default draft-folder
Signature:	To determine the user's mail signature
mailproc:	Program to post failure notices
postproc:	Program to post the message

See Also

comp(1mh), dist(1mh), forw(1mh), repl(1mh), mh-alias(5mh), post(8mh)

sh (1)

Name

sh, for, case, if, while, :, ., break, continue, cd, eval, exec, exit, export, login, read, readonly, set, shift, times, trap, umask, wait – command language

Syntax

sh [**–ceiknrstuvx**] [*arg...*]

Description

The `sh` command is a command programming language that executes commands read from a terminal or a file. See **Invocation** for the meaning of arguments to the shell.

Commands
A *simple command* is a sequence of nonblank *words* separated by blanks (a blank is a **tab** or a **space**). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0. For further information, see `execve`(2). The *value* of a simple command is its exit status if it terminates normally or 200+*status* if it terminates abnormally. For a list of status values, see `sigvec`(2).

A *pipeline* is a sequence of one or more *commands* separated by |. The standard output of each command but the last is connected by a `pipe`(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more *pipelines* separated by **;**, **&**, **&&** or || and optionally terminated by **;** or **&**. **;** and **&** have equal precedence which is lower than that of **&&** and ||, **&&** and || also have equal precedence. A semicolon causes sequential execution. An ampersand causes the preceding *pipeline* to be executed without waiting for it to finish. The symbol **&&** (||) causes the *list* following to be executed only if the preceding *pipeline* returns a zero (nonzero) value. Newlines may appear in a *list,* instead of semicolons, to delimit commands.

A *command* is either a simple command or one of the following. The value returned by a command is that of the last simple command executed in the command.

for *name* [**in** *word ...*] **do** *list* **done**
Each time a **for** command is executed, *name* is set to the next word in the **for** word list. If **in** *word ...* is omitted, **in** "$@" is assumed. Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**
A **case** command executes the *list* associated with the first pattern that matches *word*. The form of the patterns is the same as that used for file name generation.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
The *list* following **if** is executed and if it returns zero, the *list* following **then** is executed. Otherwise, the *list* following **elif** is executed and if its value is zero, the *list* following **then** is executed. Failing that, the **else** *list* is executed.

while *list* [**do** *list*] **done**

A **while** command repeatedly executes the **while** *list* and, if its value is zero, executes the **do** *list;* otherwise the loop terminates. The value returned by a **while** command is that of the last executed command in the **do** *list*. Use **until** in place of **while** to negate the loop termination test.

(*list*) Execute *list* in a subshell.

{ *list* } *list* is simply executed.

The following words are only recognized as the first word of a command and when not quoted.

if then else elif fi case in esac for while until do done { }

Command substitution

The standard output from a command enclosed in a pair of back quotes (` `) may be used as part or all of a word; trailing new lines are removed.

Parameter substitution

The character **$** is used to introduce substitutable parameters. Positional parameters may be assigned values by **set**. Variables may be set by writing

name=*value* [*name*=*value*] ...

$ {*parameter*}

A *parameter* is a sequence of letters, digits or underscores (a *name*), a digit, or any of the characters * **@ # ? - $!**. The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is a digit, it is a positional parameter. If *parameter* is * or **@** then all the positional parameters, starting with **$1**, are substituted separated by spaces. **$0** is set from argument zero when the shell is invoked.

$ {*parameter*–*word*}

If *parameter* is set, substitute its value; otherwise substitute *word.*

$ {*parameter*= *word*}

If *parameter* is not set, set it to *word;* the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

$ {*parameter* ? *word*}

If *parameter* is set, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, a standard message is printed.

$ {*parameter*+*word*}

If *parameter* is set, substitute *word;* otherwise substitute nothing.

In the above *word* is not evaluated unless it is to be used as the substituted string. (So that, for example, echo ${d-´pwd´} will only execute *pwd* if *d* is unset.)

The following *parameters* are automatically set by the shell.

#	The number of positional parameters in decimal.
-	Options supplied to the shell on invocation or by **set**.
?	The value returned by the last executed command in decimal.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following *parameters* are used but not set by the shell.

HOME
The default argument (home directory) for the **cd** command.

PATH
The search path for commands (see **execution**).

MAIL
If this variable is set to the name of a mail file, the shell informs the user of the arrival of mail in the specified file.

PS1 Primary prompt string, by default '$ '.

PS2 Secondary prompt string, by default '> '.

IFS Internal field separators, normally **space**, **tab**, and **new**line.

Blank interpretation
After parameter and command substitution, any results of substitution are scanned for internal field separator characters (those found in **$IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ´´) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File name generation
Following substitution, each command word is scanned for the characters *, **?** and **[.** If one of these characters appears, the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character **.** at the start of a file name or immediately following a /, and the character /, must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the characters enclosed. A pair of characters separated by – matches any character lexically between the pair.

Quoting.
The following characters have a special meaning to the shell and cause termination of a word unless quoted.

; & ( ) | < > new line space tab

A character may be *quoted* by preceding it with a **\. \new-line** is ignored. All characters enclosed between a pair of quote marks (´´), except a single quote, are quoted. Inside double quotes ( ) parameter and command substitution occurs and \ quotes the characters \ ´ " and **$**.

"$*" is equivalent to **"$1 $2 ..."** whereas
"$@" is equivalent to **"$1" "$2" ... .**

Prompting
When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a new line is typed and further input is needed to complete a command, the secondary prompt (**$PS2**) is issued.

Input output
Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple command or may precede or follow a *command* and are not passed on to the invoked command. Substitution occurs before *word* or *digit* is used.

< *word* Use file *word* as standard input (file descriptor 0).

> *word* Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise it is truncated to zero length.

>> *word* Use file *word* as standard output. If the file exists, output is appended (by seeking to the end); otherwise the file is created.

<< *word* The shell input is read up to a line the same as *word*, or end of file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, **\new-line** is ignored, and \ is used to quote the characters \ $ ´ and the first character of *word.*

< & *digit* The standard input is duplicated from file descriptor *digit;* see `dup`(2). Similarly for the standard output using >.

< & – The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit, the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example,

... 2>&1

creates file descriptor 2 to be a duplicate of file descriptor 1.

If a command is followed by & then the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input output specifications.

Environment

The environment is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list; see `execve`(2) and `environ`(7). The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a *parameter* for each valid name found (except IFS), giving it the corresponding value. (IFS cannot be set by the environment; it can only be set in the current shell session.) Executed commands inherit the same environment. If the user modifies the values of these *parameters* or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's *parameter* to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple command* may be augmented by prefixing it with one or more assignments to *parameters*. Thus these two lines are equivalent

```
TERM=450 cmd args
(export TERM; TERM=450; cmd args)
```

If the **–k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following prints 'a=b c' and 'c':

```
echo a=b c
set –k
echo a=b c
```

Signals
The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent. (But see also **trap.**)

Execution
Each time a command is executed, the above substitutions are carried out. Except for the special commands listed below, a new process is created and an attempt is made to execute the command with an `execve`(2).

The shell parameter **$PATH** defines the search path for the directory containing the command. Each alternative directory name is separated by a colon (:). The default path is **:/bin:/usr/bin**. If the command name contains a /, the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an *a.out* file, it is assumed to be a file containing shell commands. A subshell (that is, a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Special commands
The following commands are executed in the shell process and, except where specified, no input output redirection is permitted for such commands.

: No effect; the command does nothing.

. *file* Read and execute commands from *file* and return. The search path **$PATH** is used to find the directory containing *file*.

break [n] Exit from the enclosing **for** or **while** loop, if any. If *n* is specified, break *n* levels.

continue [n] Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified, resume at the *n*th enclosing loop.

cd [arg] Change the current directory to *arg*. The shell parameter **$HOME** is the default *arg*.

eval [arg ...] The arguments are read as input to the shell and the resulting command(s) executed.

exec [arg ...] The command specified by the arguments is executed in place of this shell without creating a new process. Input output arguments may appear and if no other arguments are given cause the shell input output to be modified.

exit [n] Causes a noninteractive shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. (An end of file will also exit from the shell.)

export [name ...] The given names are marked for automatic export to the *environment* of subsequently executed commands. If no arguments are given, a list of exportable names is printed.

login [arg ...] Equivalent to 'exec login arg ...'.

read *name* ... One line is read from the standard input; successive words of the input are assigned to the variables *name* in order, with leftover words to the last variable. The return code is 0 unless the end-of-file is encountered.

readonly [name ...] The given names are marked readonly and the values of these names may not be changed by subsequent assignment. If no arguments are given, a list of all readonly names is printed.

set [–eknptuvx [arg ...]]

–e If noninteractive, exit immediately if a command fails.

–k All keyword arguments are placed in the environment for a

command, not just those that precede the command name.

–n Read commands but do not execute them.

–t Exit after reading and executing one command.

–u Treat unset variables as an error when substituting.

–v Print shell input lines as they are read.

–x Print commands and their arguments as they are executed.

– Turn off the **–x** and **–v** options.

These flags can also be used upon invocation of the shell. The current set of flags may be found in **$–**.

Remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, and so forth. If no arguments are given, the values of all names are printed.

shift The positional parameters from **$2**... are renamed **$1**...

times Print the accumulated user and system times for processes run from the shell.

trap [arg] [n] ... The *arg* is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. If *arg* is absent, all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by invoked commands. If *n* is 0, the command *arg* is executed on exit from the shell, otherwise upon receipt of signal *n* as numbered in `sigvec`(2). The *trap* with no arguments prints a list of commands associated with each signal number.

umask [*nnn*]
The user file creation mask is set to the octal value *nnn* . For further information, see `umask`(2). If *nnn* is omitted, the current value of the mask is printed.

wait [*n*] Wait for the specified process and report its termination status. If *n* is not given, all currently active child processes are waited for. The return code from this command is that of the process waited for.

Invocation

If the first character of argument zero is –, commands are read from **$HOME/.profile**, if such a file exists. Commands are then read as described below. The following flags are interpreted by the shell when it is invoked.

–c *string* If the **–c** flag is present, commands are read from *string* .

–s If the **–s** flag is present or if no arguments remain, then commands are read from the standard input. Shell output is written to file descriptor 2.

–i If the **–i** flag is present or if the shell input and output are attached to a terminal (as told by **gtty**), then this shell is *interactive*. In this case the terminate signal SIGTERM is ignored (so that 'kill 0' does not kill an interactive shell) and the interrupt signal SIGINT is caught and ignored (so that **wait** is interruptible). For further information, see `sigvec`(2). In all cases SIGQUIT is ignored by the shell.

The remaining flags and arguments are described under the **set** command.

Restrictions

If << is used to provide standard input to an asynchronous process invoked by &, the shell becomes confused about naming the input document. A garbage file `/tmp/sh*` is created, and the shell complains about not being able to find the file by another name.

The `sh` command is not 8-bit clean. The `sh5` command is 8-bit clean.

VAX Only Restriction
If `sh` is run from another program (by the system or exec system calls) whose maximum descriptor in use is number 10, the prompt string is not printed.

Diagnostics

Errors detected by the shell, such as syntax errors cause the shell to return a nonzero exit status. If the shell is being used noninteractively, then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also **exit**).

Files

```
$HOME/.profile
/tmp/sh*
/dev/null
```

See Also

csh(1), sh5(1), test(1), execve(2), environ(7)

Name

sh5, rsh5 – shell, the standard/restricted command programming language

Syntax

sh5 [**–acefhiknrstuvx**] [*args*]
rsh5 [**–acefhiknrstuvx**] [*args*]

Description

The `sh5` program is a command line interpreter and programming language that executes commands read from a terminal or a file. The `rsh5` program is a restricted version of the standard command interpreter `sh5`. It is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See **Invocation** below for the meaning of arguments to the shell. This version of the shell is from System V Release 2. For further information about the standard Bourne shell interpreter, see `sh`(1).

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 nonblank 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. For further information, see `execve`(2). The value of a simple command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally. For a list of status values, see `signal`(3).

A pipeline is a sequence of one or more commands separated by | (or, for historical compatibility, 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. That is, 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 (nonzero) 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, then the **for** command executes the **do** *list* once for each positional parameter that is set. For further information, see **Parameter Substitution** below. Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**
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 except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly. For further information, see **File Name Generation.**

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**
A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*. Otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status. The **until** command may be used in place of **while** to negate the loop termination test.

(*list*) Execute *list* in a sub-shell.

{*list*;} Simply executes *list* from current shell.

name () {*list*;}
Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below. For further information, see **Execution.**

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (`*command*`) may be used as part or all of a word. Trailing new-lines are removed.

Parameter Substitution

The character **$** is used to introduce substitutable parameters. There are two types of parameters, positional and keyword. If parameter is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

```
name = value [ name = value ] ...
```

Pattern-matching is not performed on value. There cannot be a function and a variable with the same name.

${*parameter***}**
The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or **@**, all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter***:-***word***}**
If *parameter* is set and is non-null, substitute its value. Otherwise substitute *word*.

${*parameter***:=***word***}**
If *parameter* is not set or is null set it to *word*. The value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter***:?***word***}**
If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter***:+***word***}**
If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-'pwd'}
```

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.
- Flags supplied to the shell on invocation or by the **set** command.
? The decimal value returned by the last synchronously executed command.
$ The process number of this shell.
! The process number of the last background command invoked.

The following parameters are used by the shell:

LOGNAME
The name of the user's login account, corresponding to the login name in the user database.

HOME
The default argument (home directory) for the `cd`(1) command.

PATH
The search path for commands. For further information, see **Execution** below. The user may not change **PATH** if executing under `rsh5`.

CDPATH
The search path for the `cd`(1) 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
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by **%** and a message that will be printed when the modification time changes. The default message is **you have mail.**

PS1 Primary prompt string, by default "$ ".

PS2 Secondary prompt string, by default "> ".

IFS Internal field separators, normally **space**, **tab**, and **new-line**.

SHELL
When the shell is invoked, it scans the environment for this name. For further information, see **Environment** below. If it is found and there is an 'r' in 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**. **LOGNAME**, **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.

File Name Generation

Following substitution, each command word is scanned for the characters *, ?, and [. If one of these characters appears the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character . at the start of a file name or immediately following a /, 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 may be quoted (that is, made to stand for itself) by preceding it with a \. The pair **\new-line** is ignored. All characters enclosed between a pair of single quote marks (' '), except a single quote, are quoted. Inside double quote marks (" "), parameter and command substitution occurs and \ quotes the characters \, ‘, ", and $. **"$*"** is equivalent to **"$1 $2 ..."**, whereas **"$@"** is equivalent to **"$1" "$2" ....**

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (that is, the value of **PS2**) is issued.

Input/output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple command or may precede or follow a command and are not passed on to the invoked command. 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	The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document. Otherwise, parameter and command substitution occurs, (unescaped) **\new-line** is ignored, and \ must be used to quote the characters \, $, ‘, and the first character of *word*. If – is appended to **<<**, all leading tabs are stripped from *word* and from the document.
<&digit	Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using *>&digit*.
<&–	The standard input is closed. Similarly for the standard output using **>&–**.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit, instead of the default 0 or 1. For example:

```
... 2>&1
```

This 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
```

The first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (that is, *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*.

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.

Environment

The environment is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. For further information, see `environ`(7). The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any simple command may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd
(export TERM; TERM=450; cmd)
```

These are equivalent (as far as the execution of *cmd* is concerned).

If the **–k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c
set -k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**. Otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11. For further information, 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 `execve`(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 or between the colon delimiters anywhere else in the path list. If the command name contains a / the search path is not used. Such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary **exec** later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.

cd [*arg*] Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*. The `cd`(1) command may not be executed by `rsh5`.

echo [*arg*...]
Echo arguments. See `echo`(1sh5) 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...*]
Each given *name* is marked for automatic export to the environment of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. Function names may *not* be exported.

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. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

pwd Print the current working directory. For use and description, see `pwd`(1).

read [*name...*]
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name...*]
The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**—aefhkntuvx** [*arg...*]]

- **–a** Mark variables which are modified or created for export.
- **–e** Exit immediately if a command exits with a nonzero exit status.
- **–f** Disable file name generation
- **–h** Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
- **–k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- **–n** Read commands but do not execute them.
- **–t** Exit after reading and executing one command.
- **–u** Treat unset variables as an error when substituting.
- **–v** Print shell input lines as they are read.
- **–x** Print commands and their arguments as they are executed.
- **—** Do not change any of the flags; useful in setting **$1** to **–**.

Using + rather than – causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$–**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given the values of all names are printed.

shift [*n*]
The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

test
Evaluate conditional expressions. For usage and description, see `test`(1sh5).

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 [**–fp**] [*n*]
imposes a size limit of *n*
–f imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.
–p changes the pipe size to *n* (UNIX/RT only).
If no option is given, **–f** is assumed.

umask [*nnn*]
The user file-creation mask is set to *nnn*. For further information, see `umask`(2). 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 the specified process and report its termination status. If *n* is not given all currently active child processes are waited for and the return code is zero.

Invocation

If the shell is invoked through `execve`(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/sh5**. 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.

Rsh5 Only

The `rsh5` shell is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of `rsh5` are identical to those of `sh5`, 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.

When a command to be executed is found to be a shell procedure, `rsh5` invokes `sh5` to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands. This scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably not the login directory).

The system administrator often sets up a directory of commands (/usr/rbin) that can be safely invoked by `rsh5`. Some systems also provide a restricted editor, `red`, see `ed`(1.)

Exit Status

Errors detected by the shell, such as syntax errors, cause the shell to return a nonzero exit status. If the shell is being used noninteractively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

Restrictions

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the `cd` command with a full path name to correct this situation.

If you startup a shell using `execve`(2) with an 'r' in the *argv[0]* string, the System V shell goes into restricted mode.

Files

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

See Also

cd(1), echo(1sh5), login(1), printenv(1), pwd(1), sh(1), test(1sh5) dup(2), execve(2), fork(2), pipe(2), ulimit(2), umask(2), wait(2), signal(3), a.out(5), environ(7)

shexp(1)

Name

shexp – display password expiration information for a user

Syntax

shexp [**–q**] [*username*]

Description

The command `shexp` is used to display a user's password expiration information. The specified *username*, or logname if no *username* is supplied, is converted to a UID by searching through the passwd file. The UID is then used to look up the user's entry in the Auth Data Base. The password expiration information is then printed out in `ctime(3)` format.

```
% shexp
Expires Tue Dec  6 10:49:18 EST 1988
```

If the password has already expired the word *Expires* will be replaced with the word *Expired.* If password expiration is disabled for the particular user in question the output of `shexp` will be *Never expires.*

Options

–q Instead of displaying the expiration date and time in `ctime(3)` format, `shexp` outputs it as three decimal numbers: the minimum password lifetime, the maximum password lifetime, and the password modification time. All three numbers are displayed as they are found in the `auth` database.

Restrictions

Only the super-user may obtain information about users with UIDs other than the real UID of the invoking process.

Diagnostics

User not found in passwd data base.
There is no entry in `/etc/passwd` for the specified *username*.

Cannot stat auth file.
The `auth` database is missing (security features may not be enabled).

Insufficient privilege.
An insufficiently privileged user is asking for information about a *username* with a UID different then their current real-UID.

An exit value of 0 indicates a successful operation, any other exit status indicates an error.

Files

```
/etc/auth.[pag,dir]
/etc/passwd
```

See Also

passwd(1), getauthuid(3), auth(5)
ULTRIX Security Guide for Users and Programmers

show(1mh)

Name

show – show (list) messages

Syntax

show [+*folder*] [*msgs*] **[–draft] [–header] [–noheader] [–showproc** *program*] **[–noshowproc] [switches for** `showproc`] **[–help]**

Description

Use `show` to display the contents of the current message. You can specify alternative messages or folders by using the <`+folder`> or <`msgs`> arguments. In the following example, `show` will display the contents of message 36, in the current folder, on the screen.

```
$ show 36
```

If a folder is given, it will become the current folder. The last message shown will become the current message.

You can specify a number of messages or a range of messages using the <`msgs`> argument. If you specify more than one message; the default `showproc, more,` will prompt for a <RETURN> prior to listing each message.

Typically, the messages are listed exactly as they are, with no reformatting. A program named by the `showproc` profile component is invoked to do the listing, and any switches not recognized by `show` are passed along to that program. The default program is known as `more`. The `more` command lists each message, a page at a time. When the end of the page is reached, `more` waits for a <SPACE> or <RETURN>. If you press <RETURN>, `more` will print the next line. If you press the spacebar, `more` prints the next screen of data. At the end of the message, `more` will automatically return you to the system prompt. Press **q** to quit from more before you have finished reading the message.

Options

To override the default and the `showproc` profile component, use the `-showproc program` switch. In the following example, `show` will cause the `pr`(1) program to list the current message.

```
$ show –showproc pr
```

The `MH` command `mhl` can be used as a `showproc` to show messages in a more uniform format. See `mhl`(1mh) for more details. If you are going to use the same `showproc` all the time, it is advisable to specify it in your `.mh_profile`. You can specify any switches, that the specified show program normally takes, after the `-showproc` option. If the `-noshowproc` option is specified, `/bin/cat` is used instead of `showproc`.

The `-header` switch tells `show` to display a one-line header, which contains the folder and the message number.

If the standard output is not a terminal, no queries are made, and each file is listed with a one-line header and two lines of separation.

The command `show -draft` will list the file <mh-dir>/draft if it exists.

If the profile entry `Unseen-Sequence` is present and non-empty, then `show` will remove each of the messages shown from each sequence named by the profile entry. This is similar to the `Previous-Sequence` profile entry supported by all `MH` commands which take `msgs` or `msg` arguments.

This command has the following default settings:

+*folder* defaults to the current folder
msgs defaults to cur
`-format`
`-header`

Restrictions

The `-header` switch does not work when `msgs` expands to more than one message. If the `showproc` is `mhl`, then this problem can be circumvented by referencing the `messagename` field in the `mhl` format file.

The command `show` updates your context before showing the message. Hence `show` may mark messages as seen before you actually see them. However, this is generally not a problem, unless you are using the `unseen` messages mechanism, and you interrupt `show` while it is showing `unseen` messages.

If `showproc` is `mhl`, then `show` uses a built-in `mhl`: it does not actually run the `mhl` program. Hence, if you define your own `showproc`, do not call it `mhl` since `show` will not run it.

If `more`(1) is your showproc (the default), then avoid running `show` in the background with only its standard output piped to another process. In the following incorrect example, show will go into a `tty` input state.

```
$ show | print &
```

To avoid this problem, re-direct `show`'s diagnostic output as well.
For users of `csh`:

```
$ show |& print &
```

For users of `sh`:

```
$ show 2>&1 | print &
```

Files

$HOME/.mh_profile The user profile

Profile Components

Path:	To determine the user's MH directory
Current-Folder:	To find the default current folder
Unseen-Sequence:	To name sequences denoting unseen messages
showproc:	Program to show messages

See Also

mhl(1mh), more(1), next(1mh), pick(1mh), prev(1mh), scan(1mh)

Name

size – prints the section size of an object file

Syntax

size [-o -d -x -A -B -V] [file1 ... filen]

Description

The **size** command prints information about the *text, rdata, data, sdata, bss* and *sbss* sections of each file. The file can be an object or an archive. If you do not specify a file, **size** uses *a.out* as the default.

Options

The **–o, –x,** and **–d** options print the size in octal, hexadecimal, and decimal, respectively.

The **–A** and **–B** options specify AT&T System V style output or Berkeley (4.3BSD) style output, respectively. The version of UNIX running at your site determines the default. System V style, which is more verbose than Berkeley, dumps the headers of each section. The Berkeley version prints size information for each section, regardless of whether the file exists, and prints the total in hexadecimal and decimal.

The **–V** option prints the version of **size** that you are using.

Name

size – print program's sizes

Syntax

size [*object...*]

Description

The `size` command prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in hex and decimal, of each object-file argument. If no file is specified, `a.out` is used.

See Also

a.out(5)

Name

sleep – suspend execution for a time

Syntax

sleep *time*

Description

The `sleep` command 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
```

Restrictions

The *time* must be less than 2,147,483,647 seconds.

See Also

setitimer(2), alarm(3), sleep(3)

slocal (1mh)

Name

slocal – MH receive-mail hooks

Syntax

slocal $HOME/.maildelivery [–form *formfile*] **[switches for** *postproc*] **address ... [–help]**
/usr/new/lib/mh/rcvpack file [–help]
/usr/new/lib/mh/rcvtty [*command ...*] **[–help]**

Description

A receive-mail hook is a program that is run whenever you receive a mail message. You do not invoke the hook yourself, it is invoked on your behalf by `sendmail`, when you include the line:

```
| /usr/new/lib/mh/slocal -user $USER
```

in your `.forward` file in your home directory.

The `.maildelivery` file, which is an ordinary ASCII file, controls how local delivery is performed. This file is read by *slocal.*

The format of each line in the `.maildelivery` file is

field pattern action result string

where

field:
The name of a field that is to be searched for a pattern. This is any field in the headers of the message that might be present. In addition, the following special fields are also defined:
source: the out-of-band sender information
addr: the address that was used to cause delivery to the recipient
default: this matches *only* if the message hasn't been delivered yet
*: this always matches

pattern:
The sequence of characters to match in the specified field. Matching is case-insensitive but not RE-based.

action:
The action to take to deliver the message. This is one of

file or >:
Append the message to the file named by **string** using the standard maildrop delivery process. If the message can be appended to the file, then this action succeeds.

When writing to the file, a new field is added:

Delivery-Date: date

which indicates the date and time that the message was appended to the file.

pipe or |:

Pipe the message as the standard input to the command named by *string*, using the Bourne shell *sh* (1) to interpret the string. Prior to giving the string to the shell, it is expanded with the following built-in variables:

$(sender): the return address for the message
$(address): the address that was used to cause delivery to the recipient
$(size): the size of the message in bytes
$(reply-to): either the `Reply-To:` or `From:` field of the message
$(info): miscellaneous out-of-band information

When a process is invoked, its environment is as follows:

The user/group id's are set to recipient's id's
The working directory is the recipient's directory
The umask is 0077
The process has no /dev/tty;
The standard input is set to the message
The standard output and diagnostic output are set to /dev/null
All other file-descriptors are closed
The envariables **$USER**, **$HOME**, **$SHELL** are set appropriately
No other envariables exist.

The process is given a certain amount of time to execute. If the process does not exit within this limit, it will be terminated. The amount of time is calculated as ((size times 60) plus 300) seconds, where size is the number of bytes in the message.

The exit status of the process is consulted to determine the success of the action. An exit status of zero means that the action succeeded. Any other exit status (or abnormal termination) means that the action failed.

In order to avoid any time limitations, you might implement a process that began by *forking*. The parent would return the appropriate value immediately, and the child could continue on, doing whatever it wanted for as long as it wanted. This approach should only be used if you do not care about the outcome of the action; because the success or failure of the child process cannot be passed back to `slocal`. However, if the parent is going to return a non–zero exit status, then this approach can lead to quicker delivery into your maildrop.

qpipe or *<caret>*:

Similar to *pipe*, but executes the command directly, after built-in variable expansion, without assistance from the shell.

destroy:

This action always succeeds.

result:

Indicates how the action should be performed:

A:
Perform the action. If the action succeeded, then the message is considered delivered.

R:
Perform the action. Regardless of the outcome of the action, the message is not considered delivered.

?:
Perform the action only if the message has not been delivered. If the action succeeded, then the message is considered delivered.

The file is always read completely, so that several matches can be made and several actions can be taken. The *.maildelivery* file must be owned either by the user or by root, and must be writable only by the owner. If the *.maildelivery* file cannot be found, or does not perform an action which delivers the message, then the file `/usr/new/lib/mh/maildelivery` is read according to the same rules. This file must be owned by the root and must be writable only by the root. If this file cannot be found or does not perform an action which delivers the message, then standard delivery to the user's maildrop, /usr/spool/mail/$USER, is performed.

Arguments in the *.maildelivery* file are separated by white space or comma. Since double-quotes are honored, these characters may be included in a single argument by enclosing the entire argument in double-quotes. A double-quote can be included by preceding it with a backslash.

The following example shows how `slocal` could be used.

In this example, line-by-line comments have been extracted from the code to aid readability of the example. The line numbers would not normally be in the code; they are simply there to help you. The code fragment precedes the explanation.

```
Code Fragment
-------------
```

#field	*pattern*	*action*	*result*	*string*
1)To	mmdf2	file	A	mmdf2.log
2)From	mmdf	pipe	A	err-message-archive
3)Sender	uk-mmdf-workers	file	?	mmdf2.log
4)To	Unix	>	A	unix-news
5)addr	jpo=mmdf	|	A	mmdf-redist
6)addr	jpo=ack	|	R	"resend –r $(reply-to)"
7)From	steve	destroy	A	–
8)default	–	>	?	mailbox
9)*	–	|	R	rcvalert

Commentary
---

1) file mail with mmdf2 in the "To:" line into file mmdf2.log
2) Messages from mmdf pipe to the program err-message-archive
3) Anything with the "Sender:" address "uk-mmdf-workers"
3) file in mmdf2.log if not filed already
4) "To:" unix – put in file unix-news
5) if the address is jpo=mmdf – pipe into mmdf-redist
6) if the address is jpo=ack – send an acknowledgement copy back
7) anything from steve – destroy!

8) anything not matched yet – put into mailbox
9) always run rcvalert

Four programs are currently available, *rcvdist* (redistribute incoming messages to additional recipients), *rcvpack* (save incoming messages in a *packf* file), and *rcvtty* (notify user of incoming messages). The fourth program, *rcvstore* is described on the `rcvstore`(1mh) reference page. They all reside in the */usr/new/lib/mh/* directory.

The `rcvdist` program will resend a copy of the message to all of the addresses listed on its command line. It uses the format string facility described in `mh-format`(5mh).

The `rcvpack` program will append a copy of the message to the file listed on its command line. Its use is obsoleted by the `.maildelivery`.

The `rcvtty` program executes the named file with the message as its standard input, and gives the resulting output to the terminal access daemon for display on your terminal. If the terminal access daemon is unavailable on your system, then `rcvtty` will write the output to your terminal if, and only if, your terminal has "world-writable" permission. If no file is specified, or is bogus, etc., then the `rcvtty` program will give a one-line scan listing to the terminal access daemon.

Restrictions

For compatibility with older versions of `MH`, if `slocal` cannot find the user's `.maildelivery` file, it will attempt to execute an old-style `rcvmail` hook in the user's `$HOME` directory. In particular, it will first attempt to execute `.mh_receive` file maildrop directory user. Failing that it will attempt to execute

```
$HOME/bin/rcvmail user file sender
```

before giving up and writing to the user's maildrop.

In addition, whenever a hook or process is invoked, file-descriptor three (3) is set to the message in addition to the standard input.

Only two return codes are meaningful, others should be.

Files

/usr/new/lib/mh/mtstailor	tailor file
$HOME/.maildelivery	The file controlling local delivery
/usr/new/lib/mh/maildelivery	Rather than the standard file

soelim (1)

Name

soelim – eliminate nroff source directives from nroff input

Syntax

soelim [*file...*]

Description

The `soelim` command reads the specified files or the standard input and performs the textual inclusion implied by the `nroff` directives of the form

```
.so somefile
```

when they appear at the beginning of input lines. This is useful since programs such as `tbl` do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

Note that inclusion can be suppressed by using '' instead of '.', that is

```
 so /usr/lib/tmac.s
```

A sample usage of `soelim` would be

```
soelim exum?.n | tbl | nroff -ms | col | lpr
```

Options

– File name corresponding to standard input.

Restrictions

The format of the source commands must involve no strangeness – exactly one blank must precede and no blanks follow the file name.

See Also

colcrt(1), more(1)

Name

sort – sort file data

Syntax

sort [*options*] [**–k** *keydef*] [*+pos1* [*–pos2*]] [*file...*]

Description

The `sort` command sorts lines of all the named files together and writes the result on the standard output. The name '–' means the standard input. If no input files are named, the standard input is sorted.

Options

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

–b Ignores leading blanks (spaces and tabs) in field comparisons.

–d Sorts data according to dictionary ordering: letters, digits, and blanks only.

–f Folds uppercase to lowercase while sorting.

–i Ignore characters outside the ASCII range 040-0176 in nonnumeric comparisons.

–k *keydef* The *keydef* argument is a key field definition. The format is *field_start*, [*field_end*] [*type*], where *field_start* and *field_end* are the definition of the restricted search key, and *type* is a modifier from the option list [bdfinr]. These modifiers have the functionality, for this key only, that their command line counter-parts have for the entire record.

–n Sorts fields with numbers numerically. 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. (Note that –0 is taken to be equal to 0.) Option **n** implies option **b.**

–r Reverses the sense of comparisons.

–t*x* Uses specified character as field separator.

The notation *+pos1 –pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m.n*, optionally followed by one or more of the options **bdfinr,** where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any options are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first nonblank in the field; **b** is attached independently to *pos2*. A missing *.n* means .0; a missing *–pos2* means the end of the line. Under the **–t***x* option, fields are strings separated by *x*; otherwise fields are nonempty nonblank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

These are additional options:

–c	Checks sorting order and displays output only if out of order.
–m	Merges previously sorted data.
–o*name*	Uses specified file as output file. This file may be the same as one of the inputs.
–T	Uses specified directory to build temporary files.
–u	Suppresses all duplicate entries. Ignored bytes and bytes outside keys do not participate in this comparison.

Examples

Print in alphabetical order all the unique spellings in a list of words. Capitalized words differ from uncapitalized.

```
sort -u +0f +0 list
```

Print the password file, sorted by user id number (the 3rd colon-separated field).

```
sort -t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month day) entries. The options **–um** with just one input file make the choice of a unique representative from a set of equal lines predictable.

```
sort -um +0 -1 dates
```

Restrictions

Very long lines are silently truncated.

Diagnostics

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option **c**.

Files

/usr/tmp/stm*, /tmp/* first and second tries for temporary files

See Also

comm(1), join(1), rev(1), uniq(1)

Name

sort5 – internationalized System 5 sort and/or merge files

Syntax

sort5 [**–cmu**] [**–o**output] [**–y**kmem] [**–z**recsz] [**–X**] [**–dfiMnr**] [**–bt**x] [+pos1 [–pos2]] [*files*]

Description

The `sort5` command sorts lines of the named files together and writes the result on the standard output. The standard input is read if a hyphen (-) is used as a file name or if 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 determined by the collating sequence specified by the LC_COLLATE locale. The LC_COLLATE locale is controlled by the settings of either the LANG or LC_COLLATE environment variables. See `setlocale(3int)` for more information.

Options

The following options alter the default behavior:

–c Checks that the input file is sorted according to the ordering rules; gives no output unless the file is out of order.

–m Merges only; the input files are already sorted.

–u Suppresses all but one in each set of lines having equal keys.

–o*output*
Specifies the name of an output file to use instead of the standard output. The file may be the same as one of the inputs. Blanks between **–o** and *output* are optional.

–y*kmem*
Specifies the number of kilobytes of memory to use when sorting a file. If this option is omitted, *sort5* begins using a system default memory size, and continues to use more space as needed. If *kmem* is specified, *sort5* starts using that number of kilobytes of memory. If the administrative minimum or maximum is violated, the value of the corresponding minimum or maximum is used. Thus, **–y0** is guaranteed to start with minimum memory. By convention, **–y** (with no argument) starts with maximum memory.

–z*recsz*
Records the size of the longest line read in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted using either the **–c** or **–m** options, a system default size is used. Lines longer than the buffer size cause `sort5` to terminate abnormally. Supplying the actual number of bytes (or some larger value) in the longest line to be merged prevents abnormal termination.

–X Sorts using tags. Upon input each key is converted to a tag value which is sorted efficiently. This option makes international sorting faster but it consumes more memory since both key and tag must be stored.

The following options override the default ordering rules:

–d Specifies Dictionary order. Only letters, digits and blanks (spaces and tabs) are significant in comparisons.

–f Folds lower case letters into upper case.

–i Ignores characters outside the ASCII range 040-0176 in non-numeric comparisons.

–n Sorts an initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, by arithmetic value. The **–n** option implies the **–b** option, which tells the `sort5` command to ignore leading blanks when determining the starting and ending positions of a restricted sort key.

–r Reverses 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 at *pos2*. The characters at positions *pos1* and *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, that is a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank of a sequence of blanks acts as the field separator. The blank can be either a space or a tab. All blanks in a sequence of blanks are interpreted as a 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 is altered using the following options:

–t*x* Uses *x* as the field separator character. Although it may be included in a sort key, *x* is not considered part of a field. Each occurrence of *x* is significant (for example, *xx* delimits an empty field).

–b Ignores leading blanks when determining the starting and ending positions of a restricted sort key. If the **–b** option is specified before the first *+pos1* argument, it is applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *–pos2* argument.

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 are found to be 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:

```
sort5 +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:

```
sort5 –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:

```
sort5 –r +1.0b –1.1b infile1 infile2
```

Print the password file sorted by the numeric user ID (the third colon-separated field):

```
sort5 –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):

```
sort5 –um +2 –3 infile
```

Diagnostics

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, *sort5* appends one, prints a warning message, and continues.

Files

/usr/tmp/stm???

See Also

comm(1), join(1), uniq(1), setlocale(3int), strcoll(3int)

sortbib(1)

Name

sortbib – sort bibliographic database

Syntax

sortbib [**–s***KEYS*] *database...*

Description

The `sortbib` command sorts files of records containing *refer* key-letters by user-specified keys. Records may be separated by blank lines, or by .[and .] delimiters, but the two styles may not be mixed together. This program reads through each *database* and pulls out key fields, which are sorted separately. The sorted key fields contain the file pointer, byte offset, and length of corresponding records. These records are delivered using disk seeks and reads, so `sortbib` may not be used in a pipeline to read standard input.

By default, `sortbib` alphabetizes by the first %A and the %D fields, which contain the senior author and date. The **–s** option is used to specify new *KEYS*. For instance, **–s**ATD will sort by author, title, and date, while **–s**A+D will sort by all authors, and date. Sort keys past the fourth are not meaningful. No more than 16 databases may be sorted together at one time. Records longer than 4096 characters will be truncated.

The `sortbib` command sorts on the last word on the %A line, which is assumed to be the author's last name. A word in the final position, such as "jr." or "ed.", will be ignored if the name beforehand ends with a comma. Authors with two-word last names or unusual constructions can be sorted correctly by using the `nroff` convention "\0" in place of a blank. A %Q field is considered to be the same as %A, except sorting begins with the first, not the last, word. The `sortbib` command sorts on the last word of the %D line, usually the year. It also ignores leading articles (like "A" or "The") when sorting by titles in the %T or %J fields; it will ignore articles of any modern European language. If a sort-significant field is absent from a record, `sortbib` places that record before other records containing that field.

Options

–s*KEYS*
: Specifies new sort KEYS. For example, ATD sorts by author, title, and date.

See Also

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

Name

sortm – sort messages

Syntax

sortm [*+folder*] [*msgs*] **[–datefield** *field*] **[–verbose] [–noverbose] [–help]**

Description

The command `sortm` sorts all the messages in the current folder into chronological order according to the contents of the `Date:` fields of the messages. After sorting the messages, `sort` renumbers the messages sequentially.

You can sort the messages contained in a folder other than the current folder by specifying the *+folders* argument. If you do not want to sort all the messages in a folder, you can sort some of them by specifying the *msgs* argument. The following example would sort messages 10–30 in the folder called `+test`.

```
$ sortm +test 10-30
```

Messages which are in the folder, but not specified by *msgs*, are moved to the end of the folder.

If a folder is given, it will become the current folder.

The `-verbose` switch tells `sortm` to tell you the general actions that it is taking to place the folder in sorted order.

The `-datefield <fieldname>` switch tells `sortm` the name of the field to use when making the date comparison. If you have a special field in each message, such as `Delivery-Date:`, then the `-datefield` switch can be used to tell `sortm` which field to examine. If `sortm` encounters a message without a date-field, or if the message has a date-field that `sortm` cannot parse, then `sortm` attempts to keep the message in the same relative position. However, this does not always work; for instance, if the first message encountered lacks a date which can be parsed, then it will usually be placed at the end of the messages being sorted.

When `sortm` complains about a message which it cannot order, it complains about the message number prior to sorting.

The default settings for this command are:

+folder defaults to the current folder
msgs defaults to all
`-datefield`date
`-noverbose`

Files

$HOME/.mh_profile	The user profile

Profile Components

Path:	To determine your MH directory
Current-Folder:	To find the default current folder

See Also

folder(1mh)

Name

spell, spellin, spellout – check text for spelling errors

Syntax

spell [**–v**] [**–b**] [**–x**] [**–d** *hlist*] [*+local-file*] [**–s** *hstop*] [**–h** *spellhist*] [*file...*]

spellin [*list*]

spellout [**–d**] *list*

Description

The `spell` command collects words from the named documents, and looks them up in a spelling list. Words that are not on the spelling list and are not derivable from words on the list (by applying certain inflections, prefixes or suffixes) are printed on the standard output. If no files are specified, words are collected from the standard input.

The `spell` command ignores most `troff, tbl` and `eqn` constructions.

Two routines help maintain the hash lists used by `spell`. Both expect a set of words, one per line, from the standard input. The `spellin` command combines the words from the standard input and the preexisting *list* file and places a new list on the standard output. If no *list* file is specified, a new list is generated. The `spellout` command looks up each word from the standard input and prints on the standard output those that are missing from (or present on, with option **–d**) the hashed *list* file. For example, to verify that *hookey* is not on the default spelling list, add it to your own private list, and then use it with `spell`,

```
echo  hookey  |  spellout  /usr/dict/hlista
echo  hookey  |  spellin  /usr/dict/hlista  >  myhlist
spell  -d  myhlist <filename>
```

Options

–v	Displays words not found in spelling list with all plausible derivations from spelling list.
–b	Checks data according to British spelling. Besides preferring *centre, colour, speciality, travelled,* this option insists upon *-ise* instead of *-ize* in words like *standardise.*
–x	Precedes each word with an equal sign (=) and displays all plausible derivations.
–d *hlist*	Specifies the file used for the spelling list.
–h *spellhist*	Specifies the file used as the history file.

-s *hstop*	Specifies the file used for the stop list.
+local-file	Removes words found in *local-file* from the output of the `spell` command. The argument *local-file* is the name of a file provided by the user that contains a sorted list of words, one per line. With this option, the user can specify a list of words for a particular job that are spelled correctly.

The auxiliary files used for the spelling list, stop list, and history file may be specified by arguments following the **-d**, **-s**, and **-h** options. The default files are indicated below. Copies of all output may be accumulated in the history file. The stop list filters out misspellings (for example, thier=thy-y+ier) that would otherwise pass.

Restrictions

The coverage of the spelling list is uneven; new installations will probably wish to monitor the output for several months to gather local additions.

The `spell` command works only with ASCII text files.

Files

`/usr/dict/hlist[ab]`	hashed spelling lists, American & British, default for **-d**
`/usr/dict/hstop`	hashed stop list, default for **-s**
`/dev/null`	history file, default for **-h**
`/tmp/spell.$$*`	temporary files
`/usr/lib/spell`	

See Also

deroff(1), sed(1), sort(1), tee(1)

Name

spline – interpolate smooth curve

Syntax

spline [*option...*]

Description

The `spline` command takes pairs of numbers from the standard input as abcissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output has two continuous derivatives, and a sufficient number of points to look smooth when plotted.

Options

The following options are recognized, each as a separate argument.

–a Supplies abcissa automatically and uses specified number (next argument) for spacing. Default is 1.

–k Sets the boundary constant to the specified value (next argument). By default k = 0. For example,

$$y''_0 = ky''_1, \quad y''_n = ky''_{n-1}$$

–n Uses specified number (n) in calculating intervals between lower and upper limits. (Default n = 100.)

–p Periodically produces output (matches derivates at ends). First and last input values should normally agree.

–x Uses specified numbers (next arguments) as lower and upper limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

Restrictions

A limit of 1000 input points is enforced silently.

Diagnostics

When data is not strictly monotone in x, `spline` reproduces the input without interpolating extra points.

See Also

graph(1g), plot(1g)

Name

split – split file into smaller files

Syntax

split [*–n*] [*file [name*]]

Description

The `split` command reads *file* and writes it in *n*-line pieces (default 1000), as many as necessary, onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically. If no output name is given, **x** is default.

If no input file is given, or if – is given in its stead, then the standard input file is used.

Options

– Uses standard input.

–n Writes specified number of lines to each output file. Default is 1000.

Name

stcode – translate a hexadecimal status code value to a textual message

Syntax

stcode hex_stat_code

Description

The `stcode` command prints the textual message associated with a hexadecimal status code. This command is useful when a program produces a hexadecimal status code instead of a textual message.

The `stcode` command processes predefined status codes. No provision is currently made to add user-defined status codes to the error text database.

Examples

Translate the hexadecimal status code 1c010003:

```
# stcode 1c010003
unknown interface (network computing system/RPC runtime)
```

strextract (1int)

Name

strextract – batch string extraction

Syntax

strextract [**–p** *patternfile*] [**–i** *ignorefile*] [**–d**] [*source-program...*]

Description

The `strextract` command extracts text strings from source programs. This command also writes the string it extracts to a message text file. The message text file contains the text for each message extracted from your input source program. The `strextract` command names the file by appending .msg to the name of the input source program.

In the *source-program* argument, you name one or more source programs from which you want messages extracted. The `strextract` command does not extract messages from source programs included using the `#include` directive. Therefore, you might want a source program and all the source programs it includes on a single `strextract` command line.

You can create a patterns file (as specified by *patternfile*) to control how the `strextract` command extracts text. The patterns file is divided into several sections, each of which is identified by a keyword. The keyword must start at the beginning of a new line, and its first character must be a dollar sign ($). Following the identifier, you specify a number of patterns. Each pattern begins on a new line and follows the regular expression syntax you use in the `regex`(3) routine. For more information on the patterns file, see the `patterns` (5int) reference page.

In addition to the patterns file, you can create a file that indicates strings that `extract` ignores. Each line in this ignore file contains a single string to be ignored that follows the syntax of the `regex`(3) routine.

When you invoke the `strextract` command, it reads the patterns file and the file that contains strings it ignores. You can specify a patterns file and an ignore file on the `strextract` command line. Otherwise, the `strextract` command matches all strings and uses the default patterns file.

If `strextract` finds strings which match the `ERROR` directive in the pattern file, it reports the strings to standard error (stderr.) but does not write the string to the message file.

After running `strextract`, you can edit the message text file to remove text strings which do not need translating before running `strmerge`.

It is recommended that you use `extract` command as a visual front end to the `strextract` command rather than running `strextract` directly.

Options

–i Ignore text strings specified in *ignorefile*. By default, the `strextract` command searches for *ignorefile* in the current working directory, your home directory, and `/usr/lib/intln`.

If you omit the `-i` option, `strextract` recognizes all strings specified in the patterns file.

–p Use *patternfile* to match strings in the input source program. By default, the command searches for the pattern file in the current working directory, your home directory, and finally `/usr/lib/intln`.

If you omit the `-p` option, the `strextract` command uses a default patterns file that is stored in `/usr/lib/intln/patterns`.

–d Disables warnings of duplicate strings. If you omit the `-d` option, `strextract` prints warnings of duplicate strings in your source program.

Restrictions

Given the default pattern file, you cannot cause `strextract` to ignore strings in comments that are longer than one line.

You can specify only one rewrite string for all classes of pattern matches.

The `strextract` command does not extract strings from files include with `#include` directive. You must run the `strextract` commands on these files separately.

```
% strextract -p c_patterns prog.c prog2.c
% vi prog.msg
% strmerge -p c_patterns prog.c prog2.c
% gencat prog.cat prog.msf prog2.msf
% vi nl_prog.c
% vi nl_prog2.c
% cc nl_prog.c nl_prog2.c -li
```

In this example, the `strextract` command uses the `c_patterns` file to determine which strings to match. The input source programs are named `prog.c` and `prog2.c`.

If you need to remove any of the messages or extract one of the created strings, edit the resulting message file, `prog.msg`. Under no conditions should you add to this file. Doing so could result in unpredictable behavior.

You issue the `strmerge` command to replace the extracted strings with calls to the message catalog. In response to this command, `strmerge`, creates the source message catalogs, `prog.msf` and `prog2.msf`, and the output source programs, `nl_prog.c` and `nl_prog2.c`.

You must edit `nl_prog.c` and `nl_prog2.c` to include the appropriate `catopen` and `catclose` function calls.

The `gencat` command creates a message catalog and the `cc` command creates an executable program.

See Also

intro(3int), gencat(1int), extract(1int), strmerge(1int), regex(3), catopen(3int), patterns(5int)
Guide to Developing International Software

strings(1)

Name

strings – print ASCII strings in program

Syntax

strings [**–**|**a**] [**–o**] [*–number*] *file...*

Description

The `strings` command looks for ascii strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null.

The `strings` command is useful for identifying random object files and many other things.

Options

– or **–a**	Looks through the entire object file for ASCII strings. Default is to look only in the initialized data space.
–number	Sets the minimum string length to specified number of characters and default is 4.
–o	Precedes each string with its file offset (octal).

See Also

od(1)

Name

strip – remove symbols and relocation bits

Syntax

strip name ...

Description

The **strip** command removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This can help create space after a program has been debugged.

The **strip** command and the `ld(1)`() command with the **–s** option are equivalent.

Files

/tmp/stm? temporary file

See Also

ld(1)

Name

strip – remove symbol table and relocation bits

Syntax

strip *name...*

Description

The `strip` command removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space after a program has been debugged.

The effect of `strip` is the same as use of the **–s** option of `ld`.

Restrictions

`Strip` will not strip files with unresolved relocation information.

Files

/tmp/stm? temporary file

See Also

ld(1)

Name

strmerge – batch string replacement

Syntax

strmerge [**–m** *prefix*] [**–p** *patternfile*] [**–s** *string*] *source-program...*

Description

The `strmerge` command reads the strings specified in the message file produced by `strextract` and replaces those strings with calls to the message file in the source program to create a new source program. The new version of source program has the same name as the input source program, with the prefix nl_. For example, if the input source program is named `prog.c`, the output source program is named `nl_prog.c`. You use this command to replace hard-coded messages (text strings identified by the `strextract` command) with calls to the `catgets` function and to create a source message catalog file. The source message catalog contains the text for each message extracted from your input source program. The `strmerge` command names the file by appending .msf to the name of the input source program. For example, the source message catalog for the `prog.c` program is named `prog.msf`. You can use the source message catalog as input to the `gencat` command.

At run time, the program reads the message text from the message catalog. By storing messages in a message catalog, instead of your program, you allow the text of messages to be translated to a new language or modified without the source program being changed.

In the *source-program* argument, you name one or more source programs for which you want strings replaced. The `strmerge` command does not replace messages for source programs included using the `#include` directive. Therefore, you might want a source program and all the source programs it includes on a single `strmerge` command line.

You can create a patterns file (as specified by *patternfile*) to control how the `strmerge` command replaces text. The patterns file is divided into several sections, each of which is identified by a keyword. The keyword must start at the beginning of a new line, and its first character must be a dollar sign ($). Following the identifier, you specify a number of patterns. Each pattern begins on a new line and follows the regular expression syntax you use in the `ed` editor. For more information on the patterns file, see the `patterns`(5int) reference page.

Options

–m Add *prefix* to message numbers in the output source program and source message catalog. You can use this prefix as a mnemonic. You must process source message catalogs that contain number prefixes using the `gencat`-h option. Message numbers will be in the form:

```
<prefix><msg_num>
```

Set numbers will be in the form:

```
S_<prefix><set_num>
```

If you process your input source program with this option, the resulting source program and source message catalog may not be portable. For more information, see the *Guide to Developing International Software*.

-p Use *patternfile* to match strings in the input source program. By default, the command searches for the pattern file in the current directory, your home directory and finally `/usr/lib/intln`.

If you omit the `-p` option, the `strmerge` command uses a default patterns file that is stored in `/usr/lib/intln/patterns`.

–s Write *string* at the top of the source message catalog. If you omit the `-s` option, `strmerge` uses the string specified in the `$CATHEAD` section of the patterns file.

Restrictions

You can specify only one rewrite string for all classes of pattern matches.

The `strmerge` command does not verify if the message text file matches the source file being rewritten.

The `strmerge` command does not replace strings to files included with `#include` directive. You must run the `strmerge` command on these files separately.

Examples

The following produces a message file `prog.cat` for a program called `prog.c`.

```
% strextract -p c_patterns prog.c prog2.c
% vi prog.msg
% strmerge -p c_patterns prog.c prog2.c
% gencat prog.cat prog.msf
% vi nl_prog.c
% vi nl_prog2.c
% cc nl_prog.c nl_prog2.c -li
```

In this example, the `strextract` command uses the `c_patterns` file to determine which strings to match. The input source programs are named `prog.c` and `prog2.c`.

If you need to remove any of the messages or extract one of the created strings, edit the resulting message file, `prog.msg`. Under no conditions should you add to this file. Doing so could result in unpredictable behavior.

You issue the `strmerge` command to replace the extracted strings with calls to the message catalog. In response to this command, `strmerge` creates the source message catalogs, `prog.msf` and `prog2.msf`, and the output source programs, `nl_prog.c` and `nl_prog2.c`.

Before compiling the source programs, you must edit `nl_prog.c` and `nl_prog2.c` to include the appropriate `catopen` and `catclose` function calls.

The `gencat` command creates a message catalog and the `cc` command creates an executable program.

See Also

intro(3int), extract(1int), gencat(1int), strextract(1int), trans(1int), regex(3), catopen(3int), catgets(3int), patterns(5int)
Guide to Developing International Software

stty(1)

Name

stty – set terminal options

Syntax

stty [*option...*]

Description

The `stty` command sets or reports certain input/output characteristics of the current output terminal. Output from the `stty` program is sent to the diagnostic output (standard error). The `stty` command is used in two terminal environments. The terminal environment is determined by the setting of the terminal's line discipline. If the terminal's line discipline is set to anything other than TERMIODISC (termio line discipline), refer to the sections entitled "Non-Termio Operation" and "Non-Termio Options." If your terminal line is set to the termio line discipline, refer to the sections entitled "Termio Operation" and "Termio Options."

Note that you can use the command `stty disc` to find out the current line discipline of your terminal.

Non-termio Operation

With no argument, the `stty` command reports the speed of the terminal and the settings of the options that are different from their defaults. The following arguments report the current settings of the terminal:

all	Reports all normally used non-termio option settings.
everything	Reports all non-termio option settings.

Non-Termio Options

The option strings for terminals that are not using the termio line discipline are selected from the following set:

even	Allows even parity input.
–even	Disallows even parity input.
odd	Allows odd parity input.
–odd	Disallows odd parity input.
raw	Specifies raw mode input with no input processing (for example, erase, kill, interrupt); parity bit passed back.
–raw	Negates raw mode.
cooked	Negates raw mode.
cbreak	Makes each character available to `read`(2) as it is received; all processing other than erase and kill processing is performed.
–cbreak	Makes characters available to *read* only when new line is received.

–nl	Allows carriage return for new-line, and outputs CR-LF for carriage return or new-line.
nl	Accepts only new-line to end lines.
echo	Echoes back every character typed.
–echo	Does not echo characters.
lcase	Maps upper case to lower case.
–lcase	Does not map case.
tandem	Enables flow control. The system sends out the stop character when its internal queue is in danger of overflowing on input; it sends the start character when it is ready to accept further input.
–tandem	Disables flow control.
–tabs	Replaces tabs with spaces when printing.
tabs	Preserves tabs. This option may cause unpredictable behavior if unprintable characters, such as escape sequences, are sent to the terminal.
ek	Sets erase and kill characters to the pound sign (#) and at sign (@) respectively.
termiod	Sets line discipline to termio line discipline. Note that once the line discipline has been changed to TERMIODISC, the termio options to `stty` should be used.
disc	Reports the current line discipline.
old	Sets line discipline to the old line discipline (OTTDISC).

The following commands take a character argument *c*. You may specify u or undef instead of *c* to leave the value undefined. The two character sequence of <CTRL/x> is also interpreted as a control character, with <CTRL/?> representing delete.

erase *c*	Sets the erase character to *c*. The default is the pound sign (#), but it is often reset to <CTRL/H>.
kill *c*	Sets the kill character to *c*. The default is the at sign (@), but it is often reset to <CTRL/U>.
intr *c*	Sets the interrupt character to *c*. The default is DEL or <CTRL/?> but it is often reset to <CTRL/C>.
quit *c*	Sets the quit character to *c*. The default is <CTRL/\>.
start *c*	Sets the start character to *c*. The default is <CTRL/Q>.
stop *c*	Sets the stop character to *c*. The default is <CTRL/S>.
eof *c*	Sets the end of file character to *c*. The default is <CTRL/S>.
brk *c*	Sets the break character to *c*. The default is undefined. This character causes a wakeup.
cr0 cr1 cr2 cr3	Selects style of delay for carriage return; see `ioctl`(2) for more information.

nl0 nl1 nl2 nl3 Selects style of delay for linefeed.

tab0 tab1 tab2 Selects style of delay for tab.

ff0 ff1 Selects style of delay for form feed.

bs0 bs1 Selects style of delay for backspace.

dec Sets all modes suitable for Digital Equipment Corporation operating systems users. This command sets erase to <CTRL/?>, kill to <CTRL/U>, and interrupt to <CTRL/C>. It also sets the decctlq and newcrt options.

size Prints the display size. The format is (rows) (columns).

rows i Sets the number of rows in the display to *i*.

cols i Sets the number of columns in the display to *i*.

excl Sets line to exclusive use.

–excl Clears exclusive use status.

0 Hangs up phone line immediately.

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb
Sets terminal baud rate to the number given, if possible.

The following are unsupported terminal devices:

tty33 Sets all modes suitable for the Teletype Corporation Model 33 terminal.

tty37 Sets all modes suitable for the Teletype Corporation Model 37 terminal.

vt05 Sets all modes suitable for Digital Equipment Corporation VT05 terminal.

tn300 Sets all modes suitable for a General Electric TermiNet 300.

ti700 Sets all modes suitable for Texas Instruments 700 series terminal.

tek Sets all modes suitable for Tektronix 4014 terminal.

A teletype driver that supports the job control processing of `csh`(1) and has more functionality than the basic driver is fully described in `tty`(4). The following options apply only to it:

new Uses new driver (switching flushes typeahead). Sets line discipline to NTTYDISC.

crt Sets options for a CRT (crtbs, ctlecho and, if greater than or equal to 1200 baud, it sets crterase and crtkill.)

crtbs Echoes backspaces on erase characters.

prterase Echoes characters that have been erased.

crterase Wipes out erased characters with the following combination of keystrokes: backspace-space-backspace.

–crterase	Leaves characters visible that have been erased. Invoke this option by using the backstroke key alone.
crtkill	Wipes out input. Similar to **crterase** in how it works.
–crtkill	Echoes the line kill character and a new line on line kill.
ctlecho	Echoes control characters as a circumflex followed by the character. For example, <CTRL/X> echoes as **^X**. Type two backspaces following the EOT character (<CTRL/D>).
–ctlecho	Echoes control characters as themselves; in cooked mode EOT (<CTRL/D>) is not echoed.
decctlq	Enables a start character (normally <CTRL/Q>) to restart output when it has been suspended.
–decctlq	Enables any character that you type to restart output when it has been suspended. The start character restarts output without providing any input. This is the default.
noflsh	Suppresses flushing of input and output queues upon receipt of an interrupt signal.
–noflsh	Flushes input and output queues upon receipt of interrupt signal.
tostop	Stops background jobs if they attempt terminal output.
–tostop	Allows output from background jobs to the terminal.
tilde	Converts the tilde (~) to a backslash (\) on output.
–tilde	Suppresses conversion of the tilde (~) to a backslash (\).
flusho	Discards output usually because the user hit a <CTRL/O> (internal state bit).
–flusho	Output is not discarded.
pendin	Resubmits input that is pending after a switch from cbreak to cooked. Activated when a read becomes pending or more input arrives (internal state bit).
–pendin	Specifies that input is not pending.
litout	Sends output characters without any processing.
–litout	Does normal output processing, such as inserting delays.
autoflow	Causes the terminal multiplexer to automatically respond to start and stop characters. This functionality is only provided if the stop character is <CTRL/S> and the start character is <CTRL/Q>.
–autoflow	Uses software controlled flow control.
nohang	Does not send a hangup signal if the carrier drops. Note that the **nohang** option should be used carefully. For example, suppose that you have the nohang option in your .login file and are logged in over a modem. If the carrier drops, the next call in on this line gets your active shell.
–nohang	Sends a hangup signal to control process group when carrier drops.
pass8	Allows full eight bit ascii characters in input and output.

–pass8 Strips characters to seven bits, thus disallowing the use of eight bit ascii characters.

The following special characters are applicable only when the line discipline is set to NTTYDISC. They are not normally changed. The `stty` **new** option sets the line discipline to NTTYDISC.

susp *c* Sets the suspend process character to *c*. The default is <CTRL/Z>.

dsusp *c* Sets the delayed suspend process character to *c*. The default is <CTRL/Y>.

rprnt *c* Sets the reprint line character to *c*. The default is <CTRL/R>.

flush *c* Sets the flush output character to *c*. The default is <CTRL/O>.

werase *c* Sets the word erase character to *c*. The default is <CTRL/W>.

lnext *c* Sets the literal next character to *c*. The default is <CTRL/V>.

quote *c* Sets the quote character (for erase and kill) to *c*. The default is <CTRL/\>.

Termio Operation

This section describes the arguments and options that are used when the terminal line is set to the termio line discipline (TERMIODISC). The termio line discipline is intended for programs that use either IEEE P1003 termios, or System Five style termio. Unless noted otherwise, all options in this section are applicable to both IEEE P1003 termios or System Five termio.

With no arguments, **stty** reports the speed of the terminal and the settings of certain options.

–a Reports option settings relevant to System Five termios.

–p Reports option settings relevant to IEEE POSIX termios.

For more information about the modes listed in the first five groups below refer to `termio`(4) and `termios`(4).

Termio Options

For terminals that are using the termio line discipline, select option strings from the following set:

Control Modes

parenb (–parenb) Enables (disables) parity generation and detection.

parodd (–parodd) Selects odd (even) parity.

cs5 cs6 cs7 cs8 Select character size.

0 Hangs up phone line immediately.

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb

Sets terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)

hupcl (–hupcl) Sends (does not send) hangup signal on last close of terminal line.

cstopb (–cstopb) Uses two (one) stop bits per character.

cread (–cread) Enables (disables) the receiver.

clocal (–clocal) Assumes a line without (with) modem control.

loblk (–loblk) Blocks (does not block) output from a non-current layer. (System Five termio only)

autoflow (–autoflow)
Line operates with (without) hardware monitored flow control. (POSIX only)

Input Modes

ignbrk (–ignbrk) Ignores (does not ignore) break on input.

brkint (–brkint) Signals (does not signal) INTR on break.

ignpar (–ignpar) Ignores (does not ignore) parity errors.

parmrk (–parmrk) Marks (does not mark) parity errors.

inpck (–inpck) Enables (disables) input parity checking.

istrip (–istrip) Strips (does not strip) input characters to seven bits.

inlcr (–inlcr) Maps (does not map) NL to CR on input.

igncr (–igncr) Ignores (does not ignore) CR on input.

icrnl (–icrnl) Maps (does not map) CR to NL on input.

iuclc (–iuclc) Maps (does not map) upper-case alphabetics to lower case on input.

ixon (–ixon) Enables (disables) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (–ixany) Allows any character (only DC1) to restart output.

ixoff (–ixoff) Requests that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (–opost)
Post-processes output (does not post-process output; ignores all other output modes).

olcuc (–olcuc)
Maps (does not map) lower-case alphabetics to upper case on output.

onlcr (–onlcr)
Maps (does not map) NL to CR-NL on output.

ocrnl (–ocrnl)
Maps (does not map) CR to NL on output.

onocr (–onocr)
Does not output (outputs) CRs at column zero.

onlret (–onlret)
Performs (does not perform) the CR function on the terminal NL.

ofill (–ofill)
Uses fill characters (uses timing) for delays.

ofdel (–ofdel)
Specifies fill characters as DELs (NULs).

cr0 cr1 cr2 cr3
Selects style of delay for carriage returns.

nl0 nl1 Selects style of delay for line-feeds.

tab0 tab1 tab2 tab3
Selects style of delay for horizontal tabs.

bs0 bs1 Selects style of delay for backspaces.

ff0 ff1 Selects style of delay for form-feeds.

vt0 vt1 Selects style of delay for vertical tabs.

Local Modes

isig (–isig)	Enables (disables) the checking of characters against the special control characters INTR and QUIT.
icanon (–icanon)	Enables (disables) canonical input (ERASE and KILL processing).
xcase (–xcase)	Presents canonical (unprocessed) upper/lower-case.
echo (–echo)	Echoes (does not echo) every character typed.
echoe (–echoe)	Echoes (does 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, may be confusing on escaped characters, tabs, and backspaces.
echok (–echok)	Echoes (does not echo) NL after KILL character.
echonl (–echonl)	Echoes (does not echo) NL.
noflsh (–noflsh)	Disables (enables) flush after INTR or QUIT.
ctlech (–ctlech)	Echoes (echo control characters unchanged) control characters as ^*x* and delete as ^?. (POSIX only)
prtera (–prtera)	Echoes (does not echo) erased characters enclosed within back and forward slashes (\ /) for printing terminals. (POSIX only)

crtera (–crtera)	Wipes out (simply backspace) erased characters with backspace-space-backspace. (POSIX only)
crtkil (–crtkil)	Wipes out line (kill character and newline) with backspace-space-backspace. (POSIX ONLY)

Control Assignments

control-character c	Sets *control-character* to *c*, where *control-character* is **erase, kill, intr, quit, eof, eol, min,** or **time** (min **and** time are used with **–icanon.**

The following control characters are applicable to POSIX mode only: **susp, dsusp, rprnt, flush, werase, lnext, quote.**

If *c* is preceded by a circumflex (^), then the value used is the corresponding **CTRL** character (for example, ^d is a **CTRL-d**); ^? is interpreted as DEL and ^– is interpreted as undefined.

The **new** option sets the line discipline to NTTYDISC. Note that this changes the line discipline to be a non-termio line discipline. Once this has been done the `stty` options described in the non-termio section should be used.

Combination Modes

evenp or **parity**	Enables **parenb** and **cs7.**
oddp	Enables **parenb, cs7,** and **parodd.**
–parity, –evenp, or **–oddp**	Disables **parenb,** and set **cs8.**
nl (–nl)	Unsets (sets) **icrnl, onlcr.** In addition **–nl** unsets **inlcr, igncr, ocrnl,** and **onlret.**
lcase (–lcase)	Sets (unsets) **xcase, iuclc,** and **olcuc.**
LCASE (–LCASE)	Sets (unsets) **xcase, iuclc,** and **olcuc.**
tabs (–tabs or **tab3)**	Preserves (expands to spaces) tabs when printing.
ek	Resets ERASE and KILL characters back to normal # and **@.**
sane	Resets all modes to some reasonable values.

See Also

ioctl(2), tabs(1), tset(1), tty(4), termio(4), termios(4)

style(1)

Name

style – analyze surface characteristics of a document

Syntax

style [**–ml**] [**–mm**] [**–a**] [**–e**] [**–l** *num*] [**–r** *num*] [**–p**] [**–P**] *file...*

Description

The `style` command analyzes the surface characteristics of the writing style of a document. It reports on readability, sentence length and structure, word length and usage, verb type, and sentence openers. Because `style` runs `deroff` before looking at the text, formatting header files should be included as part of the input. The default macro package **–ms** may be overridden with the flag **–mm.** The flag **–ml,** which causes `deroff` to skip lists, should be used if the document contains many lists of non-sentences. The other options are used to locate sentences with certain characteristics.

Options

–a Displays all sentences with their length and readability index.

–e Displays all sentences that begin with an expletive.

–l *num*
Displays all sentences longer than *num*.

–ml Skips lists in document.

–mm
Overrides the default macro package **–ms.**

–P Displays parts of speech of the words in the document.

–p Displays all sentences that contain a passive verb.

–r *num*
Displays all sentences whose readability index is greater than *num*.

Restrictions

Use of non-standard formatting macros may cause incorrect sentence breaks.

See Also

deroff(1), diction(1)

Name

su – substitute a user ID

Syntax

su [*username*]

su – [*username*]

su –f [*username*]

Description

The `su` command requests the password of the specified *username*. If the correct password is given, `su` changes to that *username* without changing the current directory. The user environment is unchanged except for HOME and SHELL which are taken from the password file entry for *username*. The shell that is run is also taken from the password file entry for *username*. The new user ID stays in force until the shell exits.

If no *username* is specified, 'root' is assumed. To remind the superuser of his responsibilities, the shell substitutes '#' for its usual prompt.

Options

–f Prevents `csh`(1) from executing the .cshrc file, making `su` start up faster.

– Simulates a full login.

Diagnostics

Sorry
An invalid password was supplied for the specified *username*.

Unknown login: username
The specified *username* was not found in the passwd database.

No directory
The home directory for the *username* is not accessible at this time (only with "–" argument).

No shell
The shell specified in the passwd database entry for *username* could not be executed.

Kerberos initialization failure
Consult your system administrator.

If enhanced security features are enabled the following error messages are also possible:

Requires secure terminal
Attempt to su to UID 0 on a line that is not marked *secure* in `/etc/ttys`.

User's password has expired
Access is denied because the password for *username* is expired.

This account is disabled

Access is denied because the auth entry corresponding to *username* is marked disabled.

Files

`/usr/adm/sulog`
Log file of anyone who became **root**, with a date mark.

See Also

csh(1), sh(1), passwd(5yp), environ(7), edauth(8)
ULTRIX Security Guide for Users and Programmers

Name

sum – print object file's checksum

Syntax

sum *file...*

Description

The `sum` command calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

Diagnostics

'Read error' is indistinguishable from end of file on most devices; check the block count.

See Also

wc(1)

Name

symorder – rearrange name list

Syntax

symorder *orderlist symbolfile*

Description

The *orderlist* is a file containing symbols to be found in symbolfile, 1 symbol per line.

The *symbolfile* is updated in place to put the requested symbols first in the symbol table, in the order specified. This is done by swapping the old symbols in the required spots with the new ones. If all of the order symbols are not found, an error is generated.

This program was specifically designed to cut down on the overhead of getting symbols from /vmunix.

See Also

nlist(3)

Name

sync – update the super block

Syntax

/bin/sync

Description

The `sync` command executes the `sync` system primitive. The `sync` command can be called to insure all disk writes have been completed before the processor is halted in a way not suitably done by `reboot`(8) or `halt`(8).

See `sync`(2) for details on the system primitive.

See Also

fsync(2), sync(2), halt(8), reboot(8), update(8)

tabs (1)

Name

tabs – set tabs

Syntax

tabs [**–n**] [*terminal*]

Description

The `tabs` command sets the tabs on a variety of terminals. Various terminal names given in `term`(7) are recognized; the default is, however, suitable for most 300 baud terminals.

Options

–n Does not indent left margin. Default is to indent.

See Also

stty(1), term(7)

Name

tail – print lines from file

Syntax

tail [±[number][**lbc**][**fr**]] [file]

Description

The `tail` command copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *–number* from the end of the input. The *number* is counted in units of lines, blocks or characters, according to the appended option **l, b** or **c.** When no units are specified, counting is by lines.

Specifying **r** causes `tail` to print lines from the end of the file in reverse order. The default for **r** is to print the entire file this way. Specifying **f** causes `tail` not to quit at end of *file* , but rather to reread the file repeatedly.

Restrictions

The results of the `tail` command are unpredictable with character special files.

See Also

dd(1)

talk(1)

Name

talk, otalk – talk to another user

Syntax

talk *person* [*ttyname*]

otalk *person* [*ttyname*]

Description

The `talk` command is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on your own machine, then *person* is just the person's login name. If you wish to talk to a user on another host, then *person* is of the form :

```
host!user
 or
host.user
 or
host:user
 or
user@host
```

The form user@host is perhaps preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, it sends the message

```
Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine
```

to the user you wish to talk to. At this point, the recipient of the message should reply by typing

```
talk  your_name@your_machine
```

It doesn't matter from which machine the recipient replies, as long as his login-name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing Ctrl-L will cause the screen to be reprinted, while your erase, kill, and word kill characters will work in talk as normal. To exit, just type your interrupt character; `talk` then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the *mesg* command. At the outset talking is allowed. Certain commands, in particular `nroff` and `pr`(1) disallow messages in order to prevent messy output.

In order to use the `talk` program with machines on your network that may be running earlier versions of ULTRIX, you must initiate a session with the command `otalk` (/usr/ucb/otalk) instead of the command `talk`. You must also respond to a request from a machine running an older version of the `talk` program with the `otalk` command. See the Restrictions section.

Examples

The following example demonstrates how to use the `otalk` command. In this case, user1, whose system (system1) is running ULTRIX V2.2 initiates a session with user2, whose system (system2) is running ULTRIX V3.0. User1 types the following:

```
system1> talk user2@system2
```

The following message appears on the screen of user2:

```
Message from Talk_Daemon@system2 at 12:37 ...
talk: connection requested by user1@system1.
talk: respond with:  otalk user1@system1
```

To establish the connection user2 follows the instructions from the Talk_Daemon and types the following at the system prompt:

```
system2> otalk user1@system1
```

Restrictions

The version of `talk` released with ULTRIX V3.0 uses a protocol that is incompatible with the protocol used in earlier versions. Starting with ULTRIX V3.0, the `talk` program communicates with other machines running ULTRIX, V3.0 (and later), and machines running 4.3 BSD or versions of UNIX based on 4.3 BSD.

The `talk` command is not 8-bit clean. Typing in DEC Multinational Characters (DECMCS) causes the characters to echo as a sequence of a carets (^) followed by the character represented with its high bit cleared. This limitation makes `talk` unusable if you want to communicate using a language which has DECMCS characters in its alphabet.

Files

`/etc/hosts`	to find the recipient's machine
`/etc/utmp`	to find the recipient's tty

See Also

mail(1), mesg(1), who(1), write(1), talkd(8c)

tar(1)

Name

tar – multivolume archiver

Syntax

tar [*key*] [*name...*]

Description

The tape archiver command, `tar`, saves and restores multiple files to and from a single archive. The default archive device is `/dev/rmt0h`, but any file or device may be requested through the use of options.

The *key* is a string of characters containing at most one function letter and possibly names specifying which files to dump or restore. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

This utility supports EOT handling which allows the use of multiple media. The utility prompts for the next volume when it encounters the end of the current volume.

Keys

The function portion of the key is specified by one of the following letters:

c Create a new archive on tape, disk or file. Writing starts at the beginning of the archive instead of after the last file.

r Write the named files to the end of the archive.

t List the names of the files as they occur on the input archive.

u Add the named files to the archive if they are not there already or if they have been modified since they were last put in the archive.

x Extract the named files from the archive. If the named file matches a directory whose contents had been written into the archive, the directory is recursively extracted. The owner, modification time, and mode are restored, if possible. If no file argument is given, the entire content of the archive is extracted. Note that if multiple entries specifying the same file are in the archive, the last one overwrites all previous versions extracted.

Options

You can use one or more of the following options in addition to the letter which selects the function desired.

0...9 Substitute number for the device unit number as in `/dev/rmt#h`. The default is the high density rewind tape device number zero named `/dev/rmt0h`. The command

```
tar cv4 tar.c
```

uses device `/dev/rmt4h`.

A Use next argument as archive number with which to begin output.

B Force input and output blocking to 20 blocks/record. This option allows `tar` to work across a communications channel where the blocking may not be maintained.

D Directory output in original `tar` style.

C Use to perform a directory change prior to archiving data.

F[F] Operate in *fast mode* . When **–F** is specified, `tar` skips all SCCS directories, core files, and error files. When **–FF** is specified, `tar` also skips all `a.out` and ***.o** files.

H Help mode. Print a summary of the function keys and options.

M Next arg specifies maximum archive number to be written and prints current archive number on output line.

N No multi-archive, file splitting, or new header format on output. Output directories in previous tar format. On input, set file UID & GID from file header vs. values in `/etc/passwd` and group files.

O Include file owner & group names in verbose output (t & x functions) if present in archive header. Output warning message if owner or group name not found in `/etc/passwd` or `/etc/group` file (cru functions).

P Used to specify Posix format tapes. Unnecessary with keys other than the **c** key.

R Each named file contains a list of newline separated file names which will be added to (c function key) or extracted from (x function key) the archive.

S Output User Group Standard archive format.

V Display extended verbose information. Included are the version number of `tar`, the number of blocks used on the device, the number of blocks in a file, and the protection modes given in a format similar to the **ls –l** command. In addition to this information, **V** provides the information given by the **v** option.

b Use the next argument as the blocking factor for tape records. The default is 20 (the maximum is also 20). This option should only be used with raw magnetic tape archives (See the **f** option). The block size is determined automatically when reading tapes (**x** and **t**) keys.

d Use /dev/r????? (depending on your system) as the default device (blocking factor of 10).

f Use the next argument as the name of the archive instead of /dev/rmt0h. If the name of the file is **–** , `tar` writes to standard output or reads from standard input, whichever is appropriate. Thus, `tar` can be used as the head or tail of a filter chain. You can also use `tar` to move hierarchies. The following example shows how to move the directory *fromdir* to the directory *todir :*

```
# cd fromdir; tar cf - . | (cd todir; tar xpf -)
```

h Save a copy of the actual file on the output device under the symbolic link name, instead of placing the symbolic information on the output. The default action of `tar` is to place symbolic link information on the output device. A copy of the file itself is not saved on the output device.

i Ignore checksum errors found in the archive.

l Complain if `tar` cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.

m Do not restore the modification times. The modification time will be the time of extraction. Tar normally restores modification times of regular and special files.

o Suppress the normal directory information. On output, `tar` normally places information specifying owner and modes of directories in the archive. Former versions of `tar`, when encountering this information will give error message of the form,

```
<name>/: cannot create
```

`tar` will place information specifying owner and modes of directories in the archive.

p Restore the named files to their original modes, ignoring the present `umask`(2). Setuid and sticky bit information is also restored to the superuser.

s Next argument specifies size of archive in 512 byte blocks.

v Write the name of each file treated, preceded by the function letter, to diagnostic output. Normally, `tar` does its work silently. With the **t** function key, the verbose option provides more information about the tape entries than just their names.

```
#cd /
#tar cvf tar-out vmunix
```

Produces the output "a vmunix 1490 blocks" where 1490 is the number of 512 byte blocks in the file "vmunix".

```
#tar xvf tar-out
```

Produces the output "x vmunix, 762880 bytes, 1490 blocks" where 762880 is the number of bytes and 1490 is the number of 512 byte blocks in the file "vmunix" which was extracted.

w Print the action to be taken, followed by file name, then wait for user confirmation. If a word beginning with the letter **y** is given, the action is done. Any other input means do not do it.

Examples

To archive files from /usr/include and /etc, type:

```
# tar c -C /usr/include . -C /etc .
```

The `tar` command can properly handle blocked archives.

Restrictions

There is no way to ask for the *n th* occurrence of a file.

Tape errors are handled ungracefully.

The **u** key can be slow.

The limit on file name length is 100 characters.

There is no way to follow symbolic links selectively.

On SCSI tape devices tar (when reading) may end on one volume of a multi-volume set without prompting for the next volume. This is a very infrequent condition. The next volume should be loaded and the command issued again.

Diagnostics

Indicates bad key characters and read/write errors.
Indicates if enough memory is not available to hold the link tables.

Files

```
/dev/rmt0h
/dev/rra1a
/tmp/tar*
```

See Also

mdtar(1), mt(1), tar(5)

tarsets(1)

Name

tarsets – subset kitting command file generator

Syntax

/usr/sys/dist/tarsets [**–d**] *pathname*

Arguments

pathname

Specify the root directory for the file hierarchy containing the files to be kitted in the subset.

Description

The `tarsets` command reads subset inventory records from standard input and writes a command procedure to standard output. This command procedure contains the commands required to create subset images for the subset described in the input.

The `tarsets` command is used by the `kits` utility to produce software kits for use with the `setld` utility.

All error diagnostics are written to the file *stderr* in the current directory.

Options

–d Enable debugging. Debug trace diagnostics are written to *ts.dbg* in the current directory.

Restrictions

The output command procedure produces multiple tar files. Each tar file has a goal size of 400Kb. This is an anachronism from the days of software distribution on RX50 diskettes. The command procedure is modified automatically to produce a single subset image when `tarsets` is called from the `kits` utility.

Return Value

The exit status from the `tarsets` command is zero unless a hard link referenced in the input inventory cannot be found in the input inventory, in which case the status is 1.

Diagnostics

Invalid Record on line *n*
: The input record on line *n* is not in subset inventory format.

path1 -> *path2* link reference unresolved.
: The input record for *path1* contains a pointer to *path2* in the referent field and *path2* does not appear in the inventory. This indicates that *path2* was deleted from the inventory after being created by the *invcutter* command.

Warning: file *filename* is *n* blocks too large for diskette
: This is an obsolete message. It can be ignored.

Writing Oversized File Volume...
: This is an obsolete message. It can be ignored.

i Blocks, *j* Chars on Volume *k*
: This is an informational message. The number *j* is the number of characters in the command written to the output to produce volume *k*.

Files

stderr	Diagnostic output.
ts.dbg	Debug diagnostic output.

See Also

invcutter(1), kits(1), stl_inv(5), setld(8).

Guide to Preparing Software for Distribution on ULTRIX Systems

Name

tbl – format tables for nroff or *roff

Syntax

tbl [*files...*]

Description

The `tbl` preprocessor is used for formatting tables for `nroff` or `*roff`. The input files are copied to the standard output, except for lines between .TS and .TE command lines, which are assumed to describe tables and are reformatted.

Options

–TX Produces output without fractional line motions. Use when the destination output device or printer or post-filter cannot handle fractional line motions.

–ms Reads in `ms` macros prior to table formatting.

–mm Reads in the `mm` macros prior to table formatting, if your system has the `*roff mm` macros installed.

Examples

As an example, letting \t represent a tab (which should be typed as a genuine tab) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

Town	Household Population	
	Households	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Branchburg	1644	3.49
Bridgewater	7897	3.81
Far Hills	240	3.19

If no arguments are given, `tbl` reads the standard input, so it may be used as a filter. When `tbl` is used with `eqn` or `neqn`, the `tbl` command should be first, to minimize the volume of data passed through pipes.

See Also

nroff(1)
"Tbl - A Program to Format Tables," *ULTRIX Supplementary Documents*, Vol. I:General User

tee(1)

Name

tee – pipe output to terminal and file

Syntax

tee [**–i**] [**–a**] [*file...*]

Description

The `tee` command transcribes the standard input to the standard output and makes copies in the *files*.

Options

–a Appends input to existing files.

–i Ignores interrupts.

Name

telnet – user interface to the TELNET protocol

Syntax

telnet [*host* [*port*]]

Description

The `telnet` interface is used to communicate with another host using the TELNET protocol. If `telnet` is invoked without arguments, it enters command mode, which is indicated by the prompt, telnet>. In this mode, `telnet` accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command (see below) with those arguments.

Once a connection is opened, `telnet` enters input mode. The input mode is either character-at-a-time or line-by-line, depending on what the remote system supports. In character-at-a-time mode, text is sent to the remote host as it is typed. In line-by-line mode, text is echoed locally and only completed lines are sent to the remote host. The local-echo-character, initially ^E. turns the local echo on and off, which is useful when you want to enter passwords without them echoing to the screen.

In either mode, if the *localchars* toggle is TRUE (the default in line mode), then the user's *quit, intr,* and *flush* characters are trapped locally and sent as TELNET protocol sequences to the remote side. Options such as toggle *autoflush* and toggle *autosynch* flush previous terminal input, as in *quit* and *intr,* in addition to flushing subsequent output to the terminal until the remote host acknowledges the TELNET sequence.

To issue `telnet` commands when in input mode, precede them with the `telnet` escape character, initially the control character followed by a right bracket (^]). When in command mode, use the normal terminal editing conventions.

The following commands are available:

open *host* [*port*]
: Opens a connection to the named host. If no port number is specified, `telnet` attempts to contact a TELNET server at the default port. The host specification may be either a host name or an Internet address specified in the dot notation. For further information, see `hosts`(5) and `inet`(3n).

close
: Closes a TELNET session and returns to command mode.

quit
: Closes any open TELNET session and exits `telnet`.

z
: Suspends `telnet`. This command only works when the user is using the `csh`(1).

mode *type*
: The *type* is either *line,* for line-by-line mode, or *character,* for character-at-a-time mode. The local host asks the remote host for permission to go into one or the other mode. The remote host enters the requested mode if it is capable of it.

status
: Shows the current status of `telnet`. This includes the peer one is connected to, as well as the state of debugging.

display [*argument...*]
Displays all, or some, of the **set** and **toggle** values (see below).

? [*command*]
Accesses on-line help. With no arguments, `telnet` prints a help summary. If a command is specified, `telnet` prints the help information for that command.

send *argument(s)*
Sends one or more special character sequences to the remote host. One or more of the following arguments can be specified:

escape
Sends the current `telnet` escape character (initially the control character followed by a right bracket, ^]).

synch
Sends the TELNET SYNCH sequence. This sequence causes the remote system to discard input that was previously entered but that it has not yet read. This sequence is sent as TCP urgent data and may not work if the remote system is a 4.2 BSD system. If it does not work, a lower case r may be echoed on the terminal screen.

brk
Sends the TELNET BRK (Break) sequence, which may have significance to the remote system.

ip
Sends the TELNET IP (Interrupt Process) sequence, which causes the remote system to abort the currently running process.

ao
Sends the TELNET AO (Abort Output) sequence, which causes 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. The remote system may or may not respond.

ec
Sends the TELNET EC (Erase Character) sequence, which causes the remote system to erase the last character entered.

el
Sends the TELNET EL (Erase Line) sequence, which causes the remote system to erase the line currently being entered.

ga
Sends the TELNET GA (Go Ahead) sequence. Often this sequence has no significance to the remote system.

nop
Sends the TELNET NOP (No OPeration) sequence.

?
Prints out help information for the **send** command.

set *argument value*

Sets a `telnet` variable to a specific value. The off value turns off the function associated with the variable. The current values of variables can be displayed with the **display** command.

The following variables that can be specified:

echo

Toggles between local echoing of entered characters, and suppressing echoing of entered characters when in line-by-line mode. The value is initially ^E.

escape

Enters the `telnet` command mode when you are connected to a remote system. The value is initially the control character followed by a left bracket (^[).

interrupt

Sends a TELNET IP sequence (see **send** *ip* above) to the remote host if `telnet` is in *localchars* mode (see **toggle** *localchars* below) and the *interrupt* character is typed. The initial value for the interrupt character is the terminal's **intr** character.

quit

Sends a TELNET BRK sequence (see **send** *brk* above) to the remote host if `telnet` is in *localchars* mode (see **toggle** *localchars* below) and the *quit* character is typed. The initial value for the quit character is the terminal's **quit** character.

flushoutput

Sends a TELNET AO sequence (see **send** *ao* above) to the remote host if *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *flushoutput* character is typed. The initial value for the flush character is the terminal's **flush** character.

erase

Sends a TELNET EC sequence (see **send** *ec* above) to the remote system if *telnet* is in *localchars* mode (see **toggle** *localchars* below), **and** if *telnet* is operating in character-at-time mode. The initial value for the erase character is the terminal's **erase** character.

kill

Sends a TELNET EL sequence (see **send** *el* above) to the remote system if `telnet` is in *localchars* mode (see **toggle** *localchars* below) **and** if `telnet` is operating in character-at-a-time mode. The initial value for the kill character is the terminal's **kill** character.

eof

Sends this character to the remote system if `telnet` is operating in line-by-line mode and this character is entered as the first character on a line. The initial value of the eof character is the terminal's **eof** character.

toggle *arguments...*

Toggles (between TRUE and FALSE) flags that control how `telnet` responds to events. More than one argument may be specified and the current value of these flags can be displayed with the **display** command.

Valid arguments for the `toggle` command are the following:

localchars
: Causes the *flush*, *interrupt*, *quit*, *erase*, and *kill* characters to be recognized locally and transformed into appropriate TELNET control sequences if this flag is set to TRUE. (See **set** above). The appropriate TELNET control sequences are: *ao*, *ip*, *brk*, *ec*, and *el*, respectively. For more information see the `send` command. The initial value for this toggle is TRUE in line-by-line mode, and FALSE in character-at-a-time mode.

autoflush
: Causes the `telnet` command to not display any data on the user's terminal until the remote system acknowledges (via a TELNET *Timing Mark* option) that it recognized and processed the following TELNET sequences: *ao*, *intr*, or *quit*. Both *autoflush* and *localchars* must be TRUE for *autoflush* to work in this manner. The initial value for this toggle is TRUE if the terminal user did not specify `stty noflsh`. Otherwise it is FALSE. For further information, see `stty`(1).

autosynch
: Causes the TELNET SYNCH sequence to follow the TELNET sequence that is initiated when either the *intr* or *quit* character is typed. The *autosynch* flag works in this manner when both the *autosynch* and *localchars* are TRUE. 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
: Toggles carriage return mode. When this mode is enabled, most carriage return characters received from the remote host are mapped into a carriage return followed by a line feed. It is useful only when the remote host sends carriage returns but never line feeds. The initial value for this toggle is FALSE.

debug
: Toggles socket level debugging which is useful only to the *superuser*. The initial value for this toggle is FALSE.

options
: Toggles the display of internal `telnet` protocol processing that deals 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.

Restrictions

In line-by-line mode, the terminal's EOF character is only recognized and sent to the remote system when it is the first character on a line.

test(1)

Name

test – test conditional expression

Syntax

test *expr*
[*expr*]

Description

The `test` command evaluates the expression *expr*. If the value of *expr* is true, the `test` command returns a zero exit status; otherwise, it returns a nonzero exit status. The `test` command also returns a nonzero exit status if no arguments are specified.

Options

The following primitives are used to construct *expr*:

–r *file*	Tests if the file exists and is readable.
–w *file*	Tests if the file exists and is writable.
–f *file*	Tests if the file exists and is not a directory.
–d *file*	Tests if the file exists and is a directory.
–s *file*	Tests if the file exists and has a size greater than zero.
–t [*fildes*]	Tests if the open file, whose file descriptor number is *fildes* (1 by default), is associated with a terminal device.
–z *s1*	Tests if the length of string *s1* is zero.
–n *s1*	Tests if the length of the string *s1* is nonzero.
s1 = *s2*	Tests if the strings *s1* and *s2* are equal.
s1 != *s2*	Tests if the strings *s1* and *s2* are not equal.
s1	Tests if *s1* is not the null string.
n1 **–eq** n2	Tests if number1 equals number2.
n1 **–ge** n2	Tests if number1 is greater than or equal to number2.
n1 **–gt** n2	Tests if number1 is greater than number2.
n1 **–le** n2	Tests if number1 is less than or equal to number2.
n1 **–lt** n2	Tests if number1 is less than number2.
n1 **–ne** n2	Tests if number1 is not equal to number2.

These primitives can be combined with the following operators:

!expr	Negates evaluation of expression.
expr **–a** expr	Tests logical *and* of two expressions.
expr **–o** expr	Tests logical *or* of two expressions.
(expr...)	Groups expressions.

The **-a** operator takes precedence over the **-o** operator. Note that all the operators and flags are separate arguments to `test`. Note also that parentheses are meaningful to the Shell and must be escaped.

See Also

find(1), sh(1), test(1sh5)

test(1sh5)

Name

test – condition evaluation command

Syntax

test expr
[expr]

Description

The `test` command evaluates the expression *expr*. If the value of *expr* is true, the `test` command returns a zero exit status; otherwise, it returns a nonzero exit status. The `test` command also returns a nonzero exit status if no arguments are specified. 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 primitives can 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.

Note that all the operators and flags are separate arguments to the `test` command. Note also that parentheses are meaningful to the Shell and must be escaped.

Note

In the form of the command that uses square brackets ([ ]), instead of the word *test*, the brackets must be delimited by blanks.

See Also

find(1), sh5(1), test(1)

tftp(1c)

Name

tftp – trivial file transfer program

Syntax

tftp [*host*] [*port*]

Description

The `tftp` command provides the user interface to the Internet standard Trivial File Transfer Protocol. The program allows a user to transfer files to and from a remote network site. The remote host can be specified on the command line. If you specify the remote host on the command line `tftp` uses *host* as the default host for future transfers.

If a *port* is specified, `tftp` uses that port number instead of the standard `tftp` service port. When the user invokes the `tftp` program `tftp` enters its command interpreter and awaits instructions. The prompt tftp> is displayed on the screen.

The following commands are recognized by `tftp`:

? Displays a help message that gives a brief summary of the commands.

ascii Specifies mode ascii.

binary Specifies mode binary.

connect *host-name* [*port*]
Sets the *host* and, optionally, sets *port* for transfers. Note that the TFTP protocol does not maintain connections between transfers. Because `connect` merely remembers what host should be used for transfers instead of actually creating a connection, it is not necessary to use the `connect` command. The remote host can be specified as part of the `get` or `put` commands.

get *remote-file...* [*local-file*]
Gets a file or set of files from the specified sources. If the host has already been specified, the *source* can be in the form of a filename on the remote host. If the host has not been specified, the *source* can be a string of the form *host:file,* specifying both a host and filename at the same time. If the latter form is used, the last hostname entered becomes the default for future transfers.

mode Sets the file transfer *type* to network ASCII or binary. The default type is network ASCII.

put *local-file...* [*remote-file/directory*]
: Puts a file or set of files to the specified remote file or directory. If the remote host has already been specified, the destination can be a filename on it. If the remote host has not been specified, the destination can be a string of the form *host:filename,* specifying both a host and filename at the same time. If the latter form is used, the last hostname entered becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be a UNIX machine.

quit
: Exits the `tftp` program.

rexmt
: Sets the retransmit timer.

status
: Shows what `tftp` believes to be the current connection status.

timeout
: Set the transaction timeout.

trace
: Sets the packet trace flag.

verbose
: Sets the verbose mode flag.

Restrictions

Since the TFTP protocol does not support any authentication, files must be world read (writable) on the remote system.

Because there is no user-login validation within the TFTP protocol, the remote site should have some sort of file access restrictions in place. The exact methods are specific to each site.

tic(1)

Name

tic – terminfo compiler

Syntax

tic [**–v[n]**] [*file* ...]

Description

The `tic` command translates `terminfo` files 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 `intro`(3cur). The *file* argument contains one or more `terminfo` terminal descriptions in `terminfo`(5) source format.

The `tic` command compiles all terminfo descriptions in the given files. Each description in the file describes the capabilities of a particular terminal. When a *use=entry_name* field is discovered, `tic` duplicates the capabilities in *entry_name* for the current entry, with the exception of the capabilities that are explicitly defined in the current entry.

If the environment variable TERMINFO is set, the results are placed there instead of in `/usr/lib/terminfo`. The variable TERMINFO must be a directory pathname. The compiled results are placed in subdirectories of the directory specified by the TERMINFO environment variable.

–v[*n*] Causes `tic` to output trace information showing its progress (verbose mode). The optional integer *n* is a number from 1 to 10, inclusive, indicating the desired level of information. If *n* is greater than 1, the level of detail is increased. If *n* is omitted, the default is 1.

Restrictions

Total compiled entries cannot exceed 4096 bytes.

The *entry_name* field cannot exceed 128 bytes.

When an entry, for example *entry_name_1*, contains a *use=entry_name_2* field, any canceled capabilities in *entry_name_2* must also appear in *entry_name_1* before *use=* for these capabilities to be canceled in *entry_name_1*.

Files

`/usr/lib/terminfo/*/*` Compiled terminal capability data base

See Also

intro(3cur), terminfo(5), *Guide to X/Open Curses Screen-Handling*

Name

time – time a command

Syntax

time *command*

Description

The given command is executed; after it is complete, `time` prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on the diagnostic output stream.

The `time` is built in to `csh`(1), using a different output format.

The **time** command can be used to cause a command to be timed no matter how much CPU time it takes. Thus

```
% time cp /etc/rc /usr/bill/rc
0.0u 0.1s 0:01 8% 2+1k 3+2io 1pf+0w
% time wc /etc/rc /usr/bill/rc
  52   178  1347 /etc/rc
  52   178  1347 /usr/bill/rc
  104  356  2694 total
0.1u 0.1s 0:00 13% 3+3k 5+3io 7pf+0w
%
```

indicates that the `cp` command used a negligible amount of user time (u) and about 1/10th of a system time (s); the elapsed time was 1 second (0:01), there was an average memory usage of 2k bytes of program space and 1k bytes of data space over the cpu time involved (2+1k); the program did three disk reads and two disk writes (3+2io), and took one page fault and was not swapped (1pf+0w). The word count command `wc` on the other hand used 0.1 seconds of user time and 0.1 seconds of system time in less than a second of elapsed time. The percentage '13%' indicates that over the period when it was active the command `wc` used an average of 13 percent of the available CPU cycles of the machine.

Restrictions

Elapsed time is accurate to the second, while the CPU times are measured to the 100th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

The `time` is a built-in command to `csh`(1), with a much different syntax. This command is available as `/bin/time` to `csh` users.

See Also

csh(1)

tip(1c)

Name

tip, cu – connect to a remote system

Syntax

tip [**–v**] [**–***speed*] *system-name*
tip [**–v**] [**–***speed*] *phone-number*
cu *phone-number* [**–t**] [**–s** *speed*] [**–a** *acu*] [**–l** *line*] [**–#**]

Description

The `tip` and `cu` commands establish a full-duplex connection to another system, giving the appearance of being logged in directly on the remote cpu. Modems must be present on your system and configured into the `/etc/remote` file in order for `tip` and `cu` to work. See `uucpsetup`(8) for information on how to set up the modems.

You must have an account on the system (or equivalent) into which you wish to log in. The preferred interface is `tip`. The `cu` interface is included for those people attached to the "call UNIX" command of version 7. This manual page describes only `tip`.

Options

–# Uses specified speed (#) as baud rate.

–l Uses specified terminal line.

–v Displays all variable settings.

Typed characters are normally transmitted directly to the remote system, which does the echoing as well. A tilde (~') appearing as the first character of a line is an escape signal. The tilde escapes are:

~CTRL/D~.
Drop the connection and exit (you may still be logged in on the remote machine).

~c [*name*]
Change directory to name (no argument causes a change to your home directory).

~! Escape to a shell (exiting the shell returns you to `tip`).

~> Copy file from local to remote. The `tip` command prompts for the name of a local file to transmit.

~< Copy file from remote to local. The `tip` command prompts first for the name of the file to be sent, then for a command to be executed on the remote system.

~p *from* [*to*]
Send a file to a remote UNIX host. The **put** command causes the remote UNIX system to run the command string: cat > to, while `tip` sends it the *from* file. If the *to* file is not specified the *from* file name is used. This command is actually a UNIX specific version of the **~>** command.

~t Take a file from a remote UNIX host. As in the **put** command the *to* file

defaults to the *from* file name if it isn't specified. The remote host executes the command string cat 'from';echo ^A to send the file to `tip`.

~| Pipe the output from a remote command to a local UNIX process. The command string sent to the local UNIX system is processed by the shell.

~# Send a BREAK to the remote system. For systems which don't support the necessary `ioctl` call the break is simulated by a sequence of line speed changes and DEL characters.

~s Sets a variable. See the discussion below.

~v Displays sets as they are made.

~CTRL/Z
Stop `tip` (only available with job control).

~? Displays a summary of the tilde escapes

The `tip` utility uses the file `/etc/remote` to find how to reach a particular system and to find out how it should operate while talking to the system. Refer to `remote`(5) for a full description. Each system has a default baud rate with which to establish a connection. If this value is not suitable, the baud rate to be used may be specified on the command line, for example, **tip -300 mds.**

When `tip` establishes a connection it sends out a connection message to the remote system; the default value, if any, is defined in `/etc/remote`.

When `tip` prompts for an argument (for example, during setup of a file transfer) the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt, or an interrupt, will abort the dialogue and return you to the remote system.

The `tip` command guards against multiple users connecting to a remote system by opening modems and terminal lines with exclusive access, and by honoring the locking protocol used by `uucp`(1c).

During file transfers `tip` provides a running count of the number of lines transferred. When using the ~> and ~< commands, the *eofread and eofwrite* variables are used to recognize end-of-file when reading, and specify end-of-file when writing (see below). File transfers normally depend on tandem mode for flow control. If the remote system does not support tandem mode, *echocheck* may be set to indicate `tip` should synchronize with the remote system on the echo of each transmitted character.

When `tip` must dial a phone number to connect to a system it will print various messages indicating its actions. The `tip` command supports two methods of dialing modems. Tailored subroutines built into `tip` support the DIGITAL DN-11, DF02, DF03, DF112, DF124, and DF224 modems, the Racal-Vadic 831 auto-call modem, the Ventel 212+ modem, Racal-Vadic 3451 modem, and the Bizcomp 1031 and 1032 integral call unit/modems.

A generic dialer interface provides an alternative method to tailored subroutines for each type of modem. The generic method uses entries similar to `termcap`(5) to provide `tip` with the information needed to activate some modem and place a call. The file used by the generic dialer is `/etc/acucap` and the format of entries in this file are described in `acucap`5.

Note that the generic dialer interface is used whenever the *AT* field from an entry of `/etc/remote` matches the name field of an entry of `/etc/acucap`. If no match is found, then the tailored subroutine list is searched and will be used if that modem is supported there.

Caution

When using a DIGITAL DF112 modem, disable the "Interface Test Mode Indicate" option (set switchpack2, switch 6 to the OFF position)..

VARIABLES The `tip` command maintains a set of *variables* which control its operation. Some of these variables are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the *s* escape. The syntax for variables is patterned after `vi`(1) and `mail`(1). Supplying *all* as an argument to the **set** command displays all variables readable by the user. Alternatively, the user may request display of a particular variable by attaching a ? to the end. For example *escape?* displays the current escape character.

Variables are numeric, string, character, or boolean values. Boolean variables are set merely by specifying their name; they may be reset by prepending a ! to the name. Other variable types are set by concatenating an = and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables. Variables may be initialized at run time by placing set commands (without the ~s prefix in a file .tiprc in one's home directory). The **–v** option causes `tip` to display the sets as they are made. Certain common variables have abbreviations. The following is a list of common variables, their abbreviations, and their default values.

beautify
(bool) Discard unprintable characters when a session is being scripted; abbreviated *be*.

baudrate
(num) The baud rate at which the connection was established; abbreviated *ba*.

dialtimeout
(num) When dialing a phone number, the time (in seconds) to wait for a connection to be established; abbreviated *dial*.

echocheck
(bool) Synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted; default is *off*.

eofread
(str) The set of characters which signify an end-of-transmission during a ~< file transfer command; abbreviated *eofr*.

eofwrite
(str) The string sent to indicate end-of-transmission during a ~> file transfer command; abbreviated *eofw*.

eol
(str) The set of characters which indicate an end-of-line. The `tip` command will recognize escape characters only after an end-of-line.

escape
(char) The command prefix (escape) character; abbreviated *es*; default

value is ~.

exceptions

(str) The set of characters which should not be discarded due to the beautification switch; abbreviated *ex*; default value is \t\n\f\b.

force

(char) The character used to force literal data transmission; abbreviated *fo*; default value is ^P.

framesize

(num) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated *fr*.

host

(str) The name of the host to which you are connected; abbreviated *ho*.

prompt

(char) The character which indicates an end-of-line on the remote host; abbreviated *pr*; default value is 0 This value is used to synchronize during what data transfers. The count of lines transferred during a file transfer command is based on receipt of this character.

raise

(bool) Upper case mapping mode; abbreviated *ra*; default value is *off*. When this mode is enabled, all lower case letters will be mapped to upper case by `tip` for transmission to the remote system.

raisechar

(char) The input character used to toggle upper case mapping mode; abbreviated *rc*; default value is ^A.

record

(str) The name of the file in which a session script is recorded; abbreviated *rec;* default value is tip.record.

script

(bool) Session scripting mode; abbreviated *sc*; default is *off*. When *script* is *true*, `tip` will record everything transmitted by the remote system in the script record file specified in *record*. If the *beautify* switch is on, only printable ASCII characters will be included in the script file (those characters between 040 and 0177). The variable *exceptions* is used to indicate characters which are an exception to the normal beautification rules.

tabexpand

(bool) Expand tabs to spaces during file transfers; abbreviated *tab*; default value is *false*. Each tab is expanded to 8 spaces.

verbose

(bool) Verbose mode; abbreviated *verb*; default is *true*. When verbose mode is enabled, `tip` prints messages while dialing, shows the current number of lines transferred during a file transfer operations, and more.

SHELL

(str) The name of the shell to use for the ~! command; default value is /bin/sh.

HOME

(str) The home directory to use for the ~c command; default value is taken from the environment.

Diagnostics

Diagnostics are self-explanatory.

Files

/etc/remote	global system descriptions
/etc/phones	global phone number data base
/etc/acucap	shared autodial modem database
${REMOTE}	private system descriptions
${PHONES}	private phone numbers
~/.tiprc	initialization file.
/usr/spool/uucp/LCK..*	lock file to avoid conflicts with *uucp*

See Also

acucap(5), phones(5), remote(5), uucpsetup(8)

Name

touch – update access and modification times of a file

Syntax

touch [**–amcf**] [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 already exist. If no time is specified, the current time is used. For a more detailed explanation of how to specify the date and time, see `date`(1).

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. This utility runs under the SYSTEM_FIVE environment.

Options

–a Causes `touch` to update the access time.

–c Prevents `touch` from creating the file if it did not previously exist.

–f Attempts to force the touch in spite of read and write restrictions on a *file*.

–m Causes `touch` to update the modification time.

Note that the default setting for the `touch` command is **–am**.

Restrictions

The **–m** flag used by itself will not work on files which are accessed over NFS.

See Also

date(1), utime(2)

tr(1)

Name

tr – translate characters

Syntax

tr [**–cds**] [*string1* [*string2*]]

Description

The `tr` command copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. When *string2* is short it is padded to the length of *string1* by duplicating its last character. Any combination of the options **–cds** may be used: **–c** complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 0 through 0377 octal; **–d** deletes all input characters in *string1;* **–s** squeezes all strings of repeated output characters that are in *string2* to single characters.

In either string the notation *a–b* means a range of characters from *a* to *b* in increasing ASCII order. The backslash character (\) followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A \ followed by any other character stands for that character.

The following example creates a list of all the words in 'file1' one per line in 'file2', where a word is taken to be a maximal string of alphabetics. The second string is quoted to protect \ from the Shell. 012 is the ASCII code for newline.

```
tr -cs A-Za-z '\012' <file1 >file2
```

Options

–c Translates complements: string1 to those not in string1.

–d Deletes all characters in string1 from output.

–s Squeezes succession of a character in string1 to one in output.

Restrictions

'\0', '\00', and '\000' are equivalent for NUL character.

'\012' is treated as octal 12 and not a NUL followed by characters 1 and 2.

See Also

ed(1), ascii(7), expand(1)

Name

trace – trace system calls of programs

Syntax

trace [*options*] *cmd args...*

Description

The `trace` command with no flag arguments traces for the given *cmd* and *args* all system calls made and prints a time stamp, the PID, call and/or return values and arguments and puts its output in the file trace.dump.

Options

–f *filename*
Puts dump in file *filename.*

–z Echos arguments only.

Only one of the following option arguments can be specified at one time.

–c# Traces given PIDs and their children. Up to sixteen PIDs can be specified.

–g# Traces given groups only. Up to sixteen Group IDs can be specified.

–p# Traces given PIDs only. Up to sixteen PIDs can be specified.

–s# Traces given system calls only. Up to sixteen PIDs can be specified.

–u# Traces given UIDs only. Up to sixteen PIDs can be specified.

Examples

```
trace -f ls.dump ls -l /dev >ls.out
```

runs the cmd ls -l /dev and puts the trace in ls.dump and `ls` output in ls.out.

```
trace -f csh.trace -p $$ &
```

will trace your login shell in the background. To stop the trace just send it a termination signal (that is, kill -TERM trace_pid).

Restrictions

Due to security, no one, not even the super-user can trace anyone elses programs. This sort of negates some of the usefulness of the **-g** and **-u** flags.

The `setuid` program cannot be traced.

Only 16 numbers can be given to the **-c, -p, -g, -u,** and **-s** flags.

The kernel must be configured with the SYS_TRACE option for this command to work; otherwise, the message "Cannot open /dev/trace" is printed.

Files

/dev/trace	read only character special device for reading syscall data.
trace.dump	default file for the system call trace data.

See Also

open(2), close(2), ioctl(2), select(2), read(2), trace(5)

Name

trans – translation tool for use with source message catalogs

Syntax

trans [**–c**] [**–o** *name*] *file.msf*

Description

The `trans` command assists in the translation of source message catalogs. The command reads input from *file.msf* and writes its output to either a file named `trans.msf` or a file you name on the command line. The command displays *file.msf* in a multiple window screen that lets you simultaneously see the original message, the translated text you enter, and any messages from the `trans` command. This multiple window screen is easier to use for translating messages than a single window screen.

The top window in the multiple window screen displays the text in the message source file *file.msf*. The editor displays the current message in reverse video.

In the center window, `trans` displays a prompt that asks you to enter a translated message. You use a control key editor to move the cursor and delete text in the center window. The control key sequences are defined as follows:

Key Sequence	Meaning
CTRL/k	Display control key help
CTRL/h	Back space
CTRL/l	Forward space
CTRL/w	Back word
CTRL/f	Forward word
CTRL/e	Move to end of input
CTRL/b	Move to beginning of input
CTRL/n	Next line
CTRL/p	Previous line
CTRL/u	Delete input
CTRL/i	Insert mode (default)
CTRL/r	Replace mode
DEL	Delete previous character

If you need to span more than one line with the translated text, type a backslash (\) and press the RETURN key to enable line continuation. After you finish entering the translated text, press the RETURN key to signal that you have finished translating that message.

The bottom window displays any messages generated by `trans`. If an error occurs, `trans` prompts you to re-enter the entire line, including the message label or number.

Options

–c Display comment lines beginning with a dollar sign ($) for translation, in addition to messages.

–o Call the output file *name*. The default is output file name is `trans.msf`.

Restrictions

Your terminal must be 80 columns by 24 lines for `trans` to display its three-window screen.

You cannot interrupt a `trans` session and restart it at the point you stopped. You must complete the all the changes to a file before exiting a file.

See Also

intro(3int), extract(1int), gencat(1int), strextract(1int), strmerge(1int)
Guide to Developing International Software

Name

true, false – provide test for status values

Syntax

true
false

Description

The `true` and `false` commands are usually used in a Bourne shell script. They test for the appropriate status "true" or "false" before running (or failing to run) a list of commands.

Examples

```
while true
do
     command list
done
```

Diagnostics

The `true` command has exit status zero.

See Also

csh(1), sh(1)

tset(1)

Name

tset – set terminal mode

Syntax

tset [*options*] [**–m** [*ident*] [*test baudrate*]:type] ... [type]
reset ...

Description

The `tset` command sets up your terminal when you first log in to a UNIX system. It does terminal dependent processing such as setting erase and kill characters, setting or resetting delays, sending any sequences needed to properly initialized the terminal, and the like. It first determines the *type* of terminal involved, and then does necessary initializations and mode settings. The type of terminal attached to each UNIX port is specified in the /etc/ttys database. Type names for terminals may be found in the `termcap`(5) database. If a port is not wired permanently to a specific terminal (not hardwired) it will be given an appropriate generic identifier such as *dialup*.

In the case where no arguments are specified, `tset` simply reads the terminal type out of the environment variable TERM and re-initializes the terminal. The rest of this manual concerns itself with mode and environment initialization, typically done once at login, and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script (.profile for `sh`(1) users or .login for `csh`(1) users) it is desirable to give information about the type of terminal you will usually use on ports which are not hardwired. These ports are identified in `/etc/ttys` as *dialup* or *plugboard* or *arpanet*. To specify what terminal type you usually use on these ports, the **–m** (map) option flag is followed by the appropriate port type identifier, an optional baud rate specification, and the terminal type. (The effect is to "map" from some conditions to a terminal type, that is, to tell `tset` "If I'm on this kind of port, guess that I'm on that kind of terminal".) If more than one mapping is specified, the first applicable mapping prevails. A missing port type identifier matches all identifiers. Any of the alternate generic names given in `termcap` may be used for the identifier.

A *baudrate* is specified as with `stty`(1), and is compared with the speed of the diagnostic output (which should be the control terminal). The baud rate **test** may be any combination of: **>**, **@**, **<**, and **!**; **@** means "at" and **!** inverts the sense of the test. To avoid problems with metacharacters, it is best to place the entire argument to **–m** within "'" characters; users of `csh`(1) must also put a "\" before any "!" used here. Thus

```
tset -m 'dialup>300:adm3a' -m dialup:dw2 -m 'plugboard:?adm3a'
```

causes the terminal type to be set to an *adm3a* if the port in use is a dialup at a speed greater than 300 baud; to a *dw2* if the port is (otherwise) a dialup (that is, at 300 baud or less). (The examples given here appear to take up more than one line, for text processing reasons. When you type in real `tset` commands, you must enter them entirely on one line.) If the *type* finally determined by `tset` begins with a question mark, the user is asked if he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead.

Thus, in the above case, the user will be queried on a plugboard port as to whether they are actually using an *adm3a*.

If no mapping applies and a final *type* option, not preceded by a **–m**, is given on the command line then that type is used; otherwise the identifier found in the `/etc/ttys` database will be taken to be the terminal type. This should always be the case for hardwired ports.

It is usually desirable to return the terminal type, as finally determined by `tset`, and information about the terminal's capabilities to a shell's environment. This can be done using the – option; using the Bourne shell, `sh`(1)

```
export TERM; TERM=`tset - options...`
```

Or using the C shell, `csh`(1)

```
setenv TERM `tset - options...`
```

With `csh` it is convenient to make an alias in your .cshrc:

```
alias tset 'setenv TERM `tset - \!*`'
```

Either of these aliases allow the command

```
tset 2621
```

to be invoked at any time from your login `csh`. **Note to Bourne Shell users:** It is not possible to get this aliasing effect with a shell script, because shell scripts cannot set the environment of their parent. (If a process could set its parent's environment, none of this nonsense would be necessary in the first place.)

These commands cause `tset` to place the name of your terminal in the variable TERM in the environment. For further information, see `environ`(7).

Once the terminal type is known, `tset` engages in terminal driver mode setting. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the line-kill (full line erase)) characters, and setting special character delays. Tab and newline expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character ('#' on standard systems), the erase character is changed to BACKSPACE (Control-H).

Options

– Name of terminal is output on stndout, captured by the shell, and placed in the environment variable TERM.

–e*c* Uses the specified character as the erase character. The default is the backspace character on the terminal, usually ^H. The character *c* can either be typed directly, or entered using the hat notation used here.

–I Suppresses transmitting terminal initialization strings.

–k*c* Uses the specified character as the kill character. It is similar to **–e** but for the line kill character rather than the erase character; *c* defaults to ^X (for purely historical reasons). The kill characters is left alone if **–k** is not specified. The hat notation can also be used for this option.

–n Initializes the "new" tty driver, if applicable. On systems with the Berkeley 4BSD tty driver, specifies that the new tty driver modes should be initialized for this terminal. For a CRT, the CRTERASE and CRTKILL modes are set only if the baud rate is 1200 or greater. See `tty4` for more detail.

–Q Suppresses erase and kill character message.

If `tset` is invoked as `reset`, it will set cooked and echo modes, turn off cbreak and raw modes, turn on newline translation, and restore special characters to a sensible state before any terminal dependent processing is done. Any special character that is found to be NULL or "–1" is reset to its default value.

This is most useful after a program dies leaving a terminal in a funny state. You may have to type "<LF>reset<LF>" to get it to work since <CR> may not work in this state. Often none of this will echo.

Examples

These examples all assume the Bourne shell and use the - option. If you use `csh`, use one of the variations described above. Note that a typical use of `tset` in a .profile or .login will also use the **–e** and **–k** options, and often the **–n** or **–Q** options as well. These options have not been included here to keep the examples small. (**NOTE:** some of the examples given here appear to take up more than one line, for text processing reasons. When you type in real `tset` commands, you must enter them entirely on one line.)

At the moment, you are on a 2621. This is suitable for typing by hand but not for a .profile, unless you are always on a 2621.

```
export TERM; TERM=`tset - 2621`
```

You have an h19 at home which you dial up on, but your office terminal is hardwired and known in /etc/ttys.

```
export TERM; TERM=`tset - -m dialup:h19`
```

You have a switch which connects everything to everything, making it nearly impossible to key on what port you are coming in on. You use a vt100 in your office at 9600 baud, and dial up to switch ports at 1200 baud from home on a 2621. Sometimes you use someone elses terminal at work, so you want it to ask you to make sure what terminal type you have at high speeds, but at 1200 baud you are always on a 2621. Note the placement of the question mark, and the quotes to protect the greater than and question mark from interpretation by the shell.

```
export TERM; TERM=`tset - -m 'switch>1200:?vt100' -m
'switch<=1200:2621'
```

All of the above entries will fall back on the terminal type specified in `/etc/ttys` if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals. Your most common terminal is an adm3a. It always asks you what kind of terminal you are on, defaulting to adm3a.

```
export TERM; TERM=`tset - ?adm3a`
```

If the file `/etc/ttys` is not properly installed and you want to key entirely on the baud rate, the following can be used:

```
export TERM; TERM=`tset - -m '>1200:vt100' 2621`
```

Here is a fancy example to illustrate the power of `tset` and to hopelessly confuse anyone who has made it this far. You dial up at 1200 baud or less on a concept100, sometimes over switch ports and sometimes over regular dialups. You use various terminals at speeds higher than 1200 over switch ports, most often the terminal in your office, which is a vt100. However, sometimes you log in from the university you used to go to, over the ARPANET; in this case you are on an ALTO emulating a dm2500. You also often log in on various hardwired ports, such as the console, all of which are properly entered in `/etc/ttys`. You want your erase character set to control H, your kill character set to control U, and don't want `tset` to print the "Erase set to Backspace, Kill set to Control U" message.

```
export TERM; TERM=`tset -e -k^U -Q - -m
'switch<=1200:concept100' -m 'switch:?vt100' -m
dialup:concept100 -m arpanet:dm2500`
```

Restrictions

For compatibility with earlier versions of `tset` a number of flags are accepted whose use is discouraged:

–d type — equivalent to **–m** dialup:type

–p type — equivalent to **–m** plugboard:type

–a type — equivalent to **–m** arpanet:type

–E c — Sets the erase character to *c* only if the terminal can backspace.

– — prints the terminal type on the standard output

–r — prints the terminal type on the diagnostic output.

Files

/etc/ttys port name to terminal type mapping database
/etc/termcap terminal capability database

See Also

csh(1), sh(1), stty(1), termcap(5), ttys(5), environ(7)

tsort(1)

Name

tsort – create topological sort

Syntax

tsort [*file*]

Description

The `tsort` command produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

Diagnostics

Odd data: there is an odd number of fields in the input file.

See Also

lorder(1)

Name

tty – print current terminal name

Syntax

tty [–s]

Description

The `tty` command prints the pathname of the user's terminal unless the –s (silent) is given. In either case, the exit value is zero if the standard input is a terminal and one if it is not.

Options

–s Suppresses pathname.

Diagnostics

Prints 'not a tty' if the standard input file is not a terminal.

Name

uac – Unaligned Access Message Control

Syntax

uac [s] [p] *value*

Description

The `uac` command controls the printing of "Fixed up unaligned data access for pid nnn at pc 0xAddr" messages. The command is used to set or display the flag that controls printing the message for the system or for the parent process of the user, which is typically a shell.

Options

The following options can be used with the `uac` command:

s Set/display the current flag setting for the system.

p Set/display the current flag setting for the parent process.

Values

If *value* is present, the flag is set; otherwise, the current flag setting is displayed. The *value* can be either a zero (0), which turns the flag off, or a one (1), which turns the flag on.

Restrictions

You must be superuser to set the system flag.

Name

ul – process underscores for terminal

Syntax

ul [**–i**] [**–t** *terminal*] [*name...*]

Description

The `ul` command reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable TERM. The **–t** option overrides the terminal kind specified in the environment. The file /etc/termcap is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, `ul` degenerates to `cat`(1). If the terminal cannot underline, underlining is ignored.

The **–i** option causes `ul` to indicate underlining by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an `nroff` output stream on a crt-terminal.

Options

–i Displays underscoring on separate line containing appropriate dashes (-).

–t *terminal*
Uses type of specified terminal in place your terminal's type.

Restrictions

The `nroff` command usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

See Also

man(1), nroff(1), colcrt(1)

uname(1)

Name

uname – print name of current UNIX system

Syntax

uname [*options*]

Description

The `uname` command prints the current system name of the UNIX system on the standard output file. The command determines which system is being used.

Options

–s	Print the system name (default).
–n	Print the nodename (the nodename may be a name that the system is known by 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.

Name

unget – undo a previous get of an SCCS file

Syntax

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 ignored. If – is given as a name, the standard input is read, with each line being taken as the name of an SCCS file to be processed.

Options

Keyletter arguments apply independently to each named file.

–n Retains copy of SCCS file which normally is removed from current directory.

–r*SID* Indicates delta version number. This would have been specified by `get` as the new delta. The use of this keyletter is necessary only if two or more outstanding `gets` for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

–s Suppresses normal messages, on the standard output of the intended delta's *SID*.

Diagnostics

See `sccshelp`(1) for explanations.

See Also

delta(1), get(1), sccs(1)
Guide to the Source Code Control System

uniq(1)

Name

uniq – report repeated lines in a file

Syntax

uniq [**–udc**[**+n**][**–n**]] [*input*[*output*]]

Description

The `uniq` command reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note that repeated lines must be adjacent in order to be found. For further information, see `sort`(1).

Options

The *n* arguments specify skipping an initial portion of each line in the comparison:

–*n* Skips specified number of fields. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* Skips specified number of characters in addition to fields. Fields are skipped before characters.

–c Displays number of repetitions, if any, for each line.

–d Displays only lines that were repeated.

–u Displays only unique (nonrepeated) lines.

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.

See Also

comm(1), sort(1)

Name

uptime – display system status

Syntax

uptime [-w]

Description

The `uptime` command prints the current time, the length of time the system has been up, the number of current users, and the average number of jobs in the run queue in the last 1, 5, and 15 minute periods. It is, essentially, the first line of a `w`(1) command. The **–w** option provides the same output as the `w` command.

Files

/vmunix system name list

See Also

w(1)

users (1)

Name

users – print names of users who are logged in

Syntax

users

Description

The `users` command lists the login names of the users currently on the system in a compact, one-line format.

Files

/etc/utmp

See Also

who(1)

Name

uucp, uulog, uuname – unix to unix copy

Syntax

uucp [*option...*] *source-file... destination-file*

uulog [*option...*]

uuname [*option...*]

Description

The `uucp` command copies files named by the source-file arguments to the destination-file argument. A file name either may be a path name on your machine or may have the form

system-name!pathname

where 'system-name' is taken from a list of system names which `uucp` knows about. Shell metacharacters ?*[] appearing in the pathname part will be expanded on the appropriate system.

Pathnames may be a full pathname, a pathname preceded by *~user,* where *user* is a userid on the specified system and is replaced by that user's login directory, or anything else prefixed by the current directory.

If the result is an erroneous pathname for the remote system, the copy will fail. If the destination-file is a directory, the last part of the source-file name is used. If a simple *~user* destination is inaccessible to `uucp`, data is copied to a spool directory and the user is notified by `mail`(1).

The `uucp` command preserves execute permissions across the transmission and gives 0666 read and write permissions. For further information, see `chmod`(2).

Options

The following options are interpreted by `uucp`.

–d Creates all necessary directories for the file copy.

–c Uses the source file when copying out rather than copying the file to the spool directory.

–m Sends you mail when the copy is complete.

–n*rec*
Sends mail to the recipient.

–W Expands only local files. Normally files names are prepended with the current working directory if a full path is not specified. The *–W* tells `uucp` to expand local files only.

The `uulog` command prints a summary of `uucp` and `uux` transactions that were recorded in the file `/usr/spool/uucp/LOGFILE`.

The options cause `uulog` to print logging information:

–s*sys*
Displays information about work involving specified system.

-u*user*
Displays information about work involving specified *user*.

The `uuname` command lists the uucp names of known systems.

-l Lists local system name.

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 pathname. Ask a responsible person on the remote system to send them to you. For the same reasons, you will probably not be able to send files to arbitrary pathnames.

Restrictions

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.)

Files

/usr/spool/uucp - spool directory
/usr/lib/uucp/* - other data and program files
/etc/acucap - shared autodial modem database

See Also

mail(1), uux(1c), acucap(5)
"Uucp Implementation Description," *ULTRIX Supplementary Documentation,* Vol. III:System Manager

Name

uuencode, uudecode – encode/decode a binary file for transmission via mail

Syntax

uuencode [*file*] *remotedest* | **mail** *sys1*!*sys2*!..!*decode*
uudecode [*file*]

Description

The `uuencode` and `uudecode` commands are used to send a binary file by uucp (or other) mail. This combination can be used over indirect mail links even when `uusend`(1c) is not available.

The `uuencode` command takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the *remotedest* for recreation on the remote system.

The `uudecode` command reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user "decode" should be filtered through the `uudecode` program. This way the file is created automatically without human intervention. This is possible on the uucp network by either using `sendmail` or by making `rmail` be a link to `Mail` instead of `mail`. In each case, an alias must be created in a master file to get the automatic invocation of `uudecode`.

If these facilities are not available, the file can be sent to a user on the remote machine who can uudecode it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

Restrictions

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking `uudecode` (often `uucp`) must have write permission on the specified file.

See Also

mail(1), uucp(1c), uusend(1c), uux(1c), uuencode(5)

uuid_gen (1ncs)

Name

uuid_gen – UUID generating program

Syntax

/etc/ncs/uuid_gen [–c] [–C] [–version]

Description

The `uuid_gen` program generates Universal Unique Identifiers (UUIDs). Without options, it generates a character-string representation of a UUID. The **–c** option enables you to generate a template for Network Interface Definition Language (NIDL) files. The **–C** option enables you to generate source-code representations of UUIDs, suitable for initializing variables of type `uuid_$t`.

Options

–c	Generate a template, including a UUID attribute, for an interface definition in the C syntax of NIDL.
–C	Generate a C source-code representation of a UUID.
–version	Display the version of the Network Computing Kernel (NCK) that this `uuid_gen` belongs to but do not generate a UUID. (NCK is part of the Network Computing System (NCS) on which DECrpc is based.)

Examples

Generate a character-string representation of a UUID:

```
$ /etc/ncs/uuid_gen
34dc23469000.0d.00.00.7c.5f.00.00.00
```

Generate a template for an interface definition in the C syntax of NIDL:

```
$ /etc/ncs/uuid_gen –c
%c
[
uuid(34dc239ec000.0d.00.00.7c.5f.00.00.00),
version(1)
]
interface INTERFACENAME {

}
```

Generate a template for an interface definition in the C syntax of NIDL. Redirect the output to the file `myfile.idl`.

```
$ /etc/ncs/uuid_gen –c >myfile.idl
```

Generate a C source-code representation of a UUID:

```
$ /etc/ncs/uuid_gen –C
= { 0x34dc23af,
0xf000,
0x0000,
0x0d,
{0x00, 0x00, 0x7c, 0x5f, 0x00, 0x00, 0x00} };
```

See Also

DECrpc Programming Guide

uusend(1c)

Name

uusend – send a file to a remote host

Syntax

uusend [**-m** mode] sourcefile sys1!sys2!..!remotefile

Description

The `uusend` command sends a file to a given location on a remote system. The system need not be directly connected to the local system, but a chain of `uucp(1c)` links needs to connect the two systems.

The sourcefile can be "-", meaning to use the standard input. Both of these options are primarily intended for internal use of uusend.

The remotefile can include the ~userid syntax.

Options

-m *mode*
Specifies octal number for mode of file on the remote system. Default is mode of input file.

Restrictions

All systems along the line must have the `uusend` command available and allow remote execution of it.

Some uucp systems have a restriction where binary files cannot be the input to a `uux(1c)` command. If this exists in any system along the line, the file will show up severly distorted.

Diagnostics

If anything goes wrong any further away than the first system down the line, you will never hear about it.

See Also

uucp(1c), uuencode(1c), uux(1c)

Name

uustat – uucp status inquiry and job control

Syntax

uustat [options]

Description

The `uustat` command either displays the status of or cancels previously specified `uucp` commands, or it provides general status on `uucp` connections to other systems.

Options

–c*hour*
Removes entries older than specified hour. This option can only be executed by the user `uucp` or the super-user.

–j*all*
Reports status of all requests.

–k*jobn*
Kills specified job. The killed `uucp` request must belong to the person issuing the `uustat` command unless that person has "super-user" privilege.

–m*mch*
Reports status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local `uucp` are provided.

–o*hour*
Reports status of requests which are older than specified hour.

–s*sys*
Reports status of `uucp` requests for specified system.

–u*user*
Reports status of requests issued by specified user.

–v Invokes verbose printout option. If this option is not specified, a status code is printed with each `uucp` request.

–y*hour*
Reports status of all requests that are younger than specified hour.

When no options are given, `uustat` outputs the status of all `uucp` requests issued by the current user. Note that only one of the options **–j, –m, –k, –c,** can be used with the other options. For example, the command

```
uustat -usteve -slimbo -y63 -v
```

will print the verbose status of all `uucp` jobs that were issued by user *steve* destined for system *limbo* within the last 63 hours. The format of each job status entry is:

```
job# user destination spool_time status_time status
```

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
00001	Copy failed for unknown reasons.
00002	Permission to access local file is denied.
00004	Permission to access remote file is denied.
00010	Bad uucp command is generated.
00020	Remote system cannot create temporary file.
00040	Cannot copy to remote directory.
00100	Cannot copy to local directory.
00200	Local system cannot create temporary file.
00400	Cannot execute uucp.
01000	Copy succeeded.
02000	Copy finished, job deleted.
04000	Job is queued.

The format for the machine accessibility status entries is:

```
system  status_time   last_success_time   status
```

where

system is the system in question

status_time
is the time the last status entry was made.

last_success_time
is the last time a connection was successfully made to this system. A conversation could be ended prematurely after a successful connection.

status is a self-explanatory description of the machine status.

In the current implementation `uux` requests are not recorded in the `uustat` logging files. This implies that *mail* and *news* requests are not recorded by `uustat`.

Files

/usr/spool/uucp/	spool directory (top level)
/usr/lib/uucp/L_stat	system status file
/usr/lib/uucp/R_stat	request status file

See Also

uucp(1c)

Name

uux – unix to unix command execution

Syntax

uux [–] *command-string*

Description

The `uux` command gathers 0 or more files from various systems, executes a command on a specified system, and 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 may be one of the following:

- A pathname
- A pathname preceded by *~xxx*, where *xxx* is a userid on the specified system and is replaced by that user's login directory
- Any other syntax that is prefixed by the current directory.

For example, the following command line gets the f1 files from the usg and pwba machines, executes a `diff` command and puts the results in f1.diff in the local directory.

```
uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff"
```

When using special shell characters such as <>'!, you should either quote the entire *command-string*, or you should quote the special characters as individual arguments.

The `uux` command attempts to get all files to the execution system. If both the file and command are located on different remote sites, the file is first brought to the local system and is then transferred to the execution system.

If you want to include files as arguments to a command, but you do not want those files to be processed by `uux`, enclose the filename in parentheses. For example:

```
uux a!uucp b!/usr/file (c!/usr/file)
```

The previous example sends a `uucp` command to system `a`. The `/usr/file` is transferred from system `b` to the local system, and then is passed to system `a`. When `/usr/file` arrives at system `a` the `uucp` command executes and sends `/usr/file` to system `c`.

If the request is not allowed on the remote system, the `uux` command notifies you. This response is sent through remote mail from the remote machine.

Options

–c, -l
Do not copy local file to the spool directory for transfer to the remote machine. This is the default.

–g*grade*

Specifies the *grade* which is a single letter or number from 0 to 9, A to Z, or a to z. The highest grade is 0, the lowest grade is z. The default is A. Lower grades should be specified for high-volume jobs, such as news.

-n Sends no notification to user.

-p, -
Reads stdin.

-r Queues the job, but does not start the file transfer.

-x*debug*
Produces debugging output on stdout. The *debug* option is a number between 0 and 9. Higher numbers provide more detailed information. Debugging is permitted only for those users with read access to `L.sys`(5).

-z Notify the user if the command fails.

Warning

An installation may limit the list of commands executable on behalf of an incoming request from `uux`. Typically, a restricted site permits little other than the receipt of mail through `uux`.

Restrictions

Only the first command of a shell pipeline may have a system-name!. All other commands are executed on the system of the first command.

The use of the shell metacharacter asterisk (*) shell metacharacter may not behave as you expect. The shell tokens (<< >>) are not implemented.

You are not notified when execution on a remote machine is denied. Only commands listed in `/usr/lib/uucp/L.cmds` on the remote system are executed at the remote system.

Files

/usr/spool/uucp spool directory
/usr/lib/uucp/* other data and programs

See Also

uucp(1c)
"Uucp Implementation Description" *ULTRIX Supplementary Documents* Vol. III: System Manager

Name

val – validate SCCS file

Syntax

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 that begin with a "-" and named files.

The `val` command 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.

The `val` command 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.

Options

The effects of any keyletter argument apply independently to each named file on the command line. The keyletter arguments are defined as follows:

– Causes stdin to be read until end of file.

–s Suppresses all error messages.

–r*SID*
Indicates specified delta version 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, and so forth, 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.

–m*name*
Compares specified value with the SCCS val.1 keyword.

–y*type*
Compares specified type with SCCS keyword.

The 8-bit code returned by `val` is a disjunction of the possible errors. It can be interpreted as a bit string where set bits are interpreted (from left to right) as:

bit 0 = missing file argument
bit 1 = unknown or duplicate keyletter argument
bit 2 = corrupted SCCS file
bit 3 = can't 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 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.

Restrictions

The `val` command can process up to 50 files on a single command line. Any number above 50 produces a core dump.

Diagnostics

Use `sccshelp`(1) for explanations.

See Also

admin(1), delta(1), get(1), prs(1), sccs(1)
Guide to the Source Code Control System

Name

vc – version control program

Syntax

vc [**–a**] [**–t**] [**–c***char*] [**–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 standard output is conditional. It is 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 noncontrol character are copied in their entirety.

A keyword is composed of 9 or fewer 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 should contain blanks or tabs.

Replacement of keywords by values occurs 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 \.

Options

Keyletter arguments:

–a Replaces the keywords surrounded by control characters in all text lines.

–c*char*
Specifies a control character to be used in place of **:**.

–s Suppresses all warning messages.

–t Ignores all characters from the beginning of the line to the first tab character. If one is found, all characters up to and including the *tab* are discarded.

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, fB: 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

= != > < all of equal precedence

&

|

Parentheses can 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. The `vc` command halts execution, and returns an exit code of 1.

Diagnostics

Use `help`(1) for explanations.

Exit Codes

0 – normal
1 – any error

Name

vcc – VAX C compiler

Syntax

vcc [*option...*] *file...*

Description

The `vcc` command invokes the VAX C compiler for ULTRIX and accepts the following types of arguments:

- Arguments whose names end with .c. These arguments are treated as C source programs. The source code is compiled and the resulting object code is left in a file whose name is the same as the source except with a .o file extension. If you choose to compile and load a single program in one step, the `vcc` command program deletes the intermediate object code file.
- Arguments whose names end with .s. These arguments are treated as assembly source programs and are passed to the assembler, which creates an output file with a .o extension.
- Arguments whose names end in something other than .c or .s. These arguments are treated as either compiler or linker option arguments or C-compatible object programs that were produced during an earlier `vcc` compilation or were extracted from the libraries of C-compatible routines.

Options

The VAX C compiler for ULTRIX can produce two types of object files: the standard BSD .o format used by `ld`(1), or an object format that can be read only by `lk`(1). The object file format, and the linker used is controlled by the **–V** lk-object option. By default, the compiler produces standard BSD .o format.

The following options are accepted by the `vcc` command. See `lk`(1) or `ld`(1) for load-time options.

–b Does not pass the **–lc** library to the linker by default.

–B*string* Finds substitute compiler, preprocessor, assembler, and linker in the files named by *string*. If *string* is empty, uses a standard backup version.

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

–D*name=def*
–D*name* Defines *name* to the preprocessor. This functions as if an additional #define preprocessor directive were embedded in the source code. If no definition is given, the name is defined as 1.

–E Runs only the macro preprocessor on the named C programs and sends the result to the standard output.

–Em Runs only the macro preprocessor on the named C programs

and produces the makefile dependencies.

–f Uses single-precision rather than double-precision floating point representation. Procedure arguments are still promoted to double-precision floating point format. Programs with a large number of single-precision computations run faster with this option. However, a slight loss in precision may result since intermediate results are saved using a single-precision representation rather than the default double-precision representation.

–g Generates additional symbol table information for `dbx`(1). This also passes the **–lg** flag to the linker.

–I*dir* Seeks #include files whose names do not begin with a directory specification in the following directories: first, in the directory of the *file* argument; second, in directories named in **–I** options; finally, in directories in a standard list.

–l*x* Uses the specified library. This option lists an abbreviation for the library name `/lib/lib`*x*.a, where *x* is a string. If the library is not found, the linker tries `/usr/lib/lib`*x*.a. If that does not exist, the linker tries `/usr/local/lib/lib`*x*.a. A search for a library starts when the library name is encountered, so the placement of a **–l** within the compilation or the linker command line is significant.

–Md Specifies DFLOAT (the default) double-precision floating point type and passes the **–lc** flag to the linker.

–Mg Specifies GFLOAT double precision floating point type and passes the **–lcg** flag to the linker. This option uses the GFLOAT version of **libc**. If the math library is used with code compiled with the **–Mg** flag, the GFLOAT version is linked by specifying **–lmg** to the `vcc` or the linker command.

–o *output* Names the final output file *output*. If this option is used, the file a.out is left undisturbed. If the named file has a .o or .a file extension, the following error message is displayed: **–o would overwrite.**

–O Invokes the object-code improver. This option is on by default and has no effect.

–p Produces code that counts the number of times each routine is called. If loading takes place, it replaces the standard startup routine with one that initially calls `monitor` and writes out a mon.out file upon normal termination of program execution. The `prof` command is then used to generate an execution profile.

–pg Produces counting code similar to that generated by **–p,** but invokes a run-time recording mechanism that keeps more extensive statistics and produces a gmon.out file upon normal termination. In addition, a profiling library is

searched instead of the standard C library. The `gprof` command is then used to generate an execution profile.

-t [p0al] Finds only the designated preprocessor, compiler, assembler, and linker in the files whose names are constructed by a **-B** option.

-U*name* Removes any initial definition of *name*.

-v *file* Produces a listing in *file*, complete with cross-reference and machine code listing sections.

-V *arg* Compiles the source code using vendor specific options. The available options are described in detail in the *Guide to VAX C for ULTRIX*.

The following is a list of the available options:

cross_reference	Generates a cross reference listing section
debug	Generates a loadable module for use with dbx
define	Assigns a specified value to a name
g_float	Uses the G_floating point type
list	Generates a list file
lk_object	Generates object files in lk format, instead of BSD .o format and uses the lk linker
machine_code	Generates the machine code listing section
object	Generates an object file with a specific name
optimize	Selects code optimization
show	Includes symbol and intermediate expansions
standard	Selects portability mode
undefine	Revokes the assignment of a value to a name
warnings	Disables warning or informational messages

-w Suppresses warning diagnostics.

-Y*[option]* Compiles a file for one of the following options:

SYSTEM_FIVE

BSD

POSIX

If no **–Y** option is specified, `vcc` searches for the PROG_ENV variable to be defined. If PROG_ENV is set to SYSTEM_FIVE or POSIX, the effect is the same as **–YSYSTEM_FIVE** or **–YPOSIX**. If PROG_ENV is not set to either SYSTEM_FIVE or POSIX, the effect is the same as **–YBSD**.

If no option is specified with **–Y**, the default is **–YSYSTEM_FIVE**. If an option other than SYSTEM_FIVE, BSD, or POSIX is specified, a warning message is printed and **–Y** is ignored. If there are multiple **–Y** options, only the last one takes effect.

If **–YSYSTEM_FIVE** is explicitly specified, the **–YSYSTEM_FIVE** parameter is added to the linker call. In addition, the following occurs:

- **–DSYSTEM_FIVE** is added to the `vaxc` command (or `cpp` command if **–E** is specified).
- The linker parameters **–lc**, **–lcg**, or **–lc_p** are preceded with **–lcV**, **–lcVg**, or **–lcV_p** (if not suppressed by **–b**).
- The linker parameters **–lm**, **–lmg**, or **–lmp** are changed to **–lmV**, **–lmVg**, or **–lmV_p** (if present).

If **–YBSD** is specified, then the parameter **–YBSD** is added to the `lk` call.

If **–YPOSIX** is specified, then the parameter **–DPOSIX** is added to the `vaxc` call. Also, the parameter **–YPOSIX** is added to the linker call.

If **–Y** does not exist and PROG_ENV is not defined, the default is **–YSYSTEM_FIVE**.

Default Symbols And Macros

The VAX C compiler recognizes the following predefined symbols. The symbols are all assigned the value one (1). You can use these symbols to separate portable and nonportable code within your VAX C programs:

vaxc	VAXC
vax11c	VAX11
vax	VAX

In addition to the VAX symbols definitions, listed above, the VAX C compiler for ULTRIX provides the following default symbols:

unix	Any UNIX system
bsd4_2	Berkeley UNIX Version 4.2
ultrix	ULTRIX only
vax	VAX processor only

The VAX C compiler recognizes the following predefined macros:

__DATE__	Evaluates to a string, specifying the compilation date
__FILE__	Evaluates to a string, specifying the current source file
__LINE__	Evaluates to an integer, specifying the line containing the macro reference.
__TIME__	Evaluates to a string, specifying the compilation time

Restrictions

The compiler treats the register keyword as a suggestion, attaching the register keyword to a variable declaration does not guarantee that the variable will be allocated to a register.

If the **–Mg** flag is used to produce GFLOAT code, it must be used when compiling all of the modules to be linked. Use the **–Mg** flag if you use the `vcc` command to invoke the linker indirectly to link the modules. If you invoke the linker directly, use the **–lcg** flag rather than **–lc flag.** If the math library is used, specify the **–lmg** flag rather than the **–lm** flag in order to use the GFLOAT version.

The compiler and the linker, cannot detect the use of mixed double floating point types. If you use them, your program's results may be erroneous.

Diagnostics

The diagnostics produced by VAX C are self-explanatory. Occasional messages are produced by the assembler or loader.

Files

file.c	input file
file.o	object file
a.out	loaded output
/usr/bin/vcc	command program
/lib/cpp	preprocessor
/usr/lib/cerrfile	error message file
/usr/lib/vaxc	compiler
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/usr/lib/gcrt0.o	startoff for gprof-profiling
/lib/libc.a	standard library
/usr/libcg.a	GFLOAT version of the standard library
/usr/lib/libc_p.a	profiling library
/usr/include	standard directory for #include files
/usr/man/man1/vcc.1	manual page
mon.out	file produced for analysis by prof(1)
gmon.out	file produced for analysis by gprof(1)

See Also

adb(1), as(1), dbx(1), gprof(1), ld(1), lk(1), prof(1), monitor(3)
Guide to VAX C

Name

vdoc – invokes CDA Viewer for character-cell displays

Syntax

vdoc [-f *format*] [-O *options_file*] [-r] [-w *paper_width*] [-h *paper_height*] [-p] *inputfile*

Description

The `vdoc` command invokes the CDA Viewer that enables you to view the *inputfile* on a character-cell terminal. If *inputfile* is not specified, `vdoc` reads from standard input.

Options

–f *format*

Specifies the format of *inputfile* and invokes an appropriate input converter as part of CDA. The ddif, dtif and text input converters are provided in the base system kit. Additional converters can be added by the CDA Converter Library and other layered products. Contact your system manager for a complete list of the input formats supported on your system. The default format is ddif.

–O *options_file*

Names the file passed to the input converter to control specific processing options in that converter. Refer to your documentation set for a description of converter options.

The options file has a default file type of .cda_options. Each line of the options file specifies a format name that can optionally be followed by _input or _output to restrict the option to either an input or output converter. The second word is a valid option preceded by one or more spaces, tabs, or a slash (/) and can contain upper- and lowercase letters, numbers, dollar signs, and underlines. The case of letters is not significant. If an option requires a value, then spaces, tabs, or an equal sign can separate the option from the value.

Each line can optionally be preceded by spaces and tabs and can be terminated by any character other than those that can be used to specify the format names and options. The syntax and interpretation of the text that follows the format name is specified by the supplier of the front and back end converters for the specified format.

To specify several options for the same input or output format, specify one option on a line. If an invalid option for an input or output format or an invalid value for an option is specified, the option may be ignored or an error message may be returned. Each input or output format that supports processing options specifies any restrictions or special formats required when specifying options.

By default, any messages that occur during processing of the

options file are written to the system *standard error location.* For those input formats that support a LOG option, messages can be directed to a log file.

–r Specifies that the CDA Viewer is to override the format of the document. If the -r qualifier is not specified, the CDA Viewer retains the formatting information stored in the document.

–w *paper_width* Specifies the paper width in units of characters. The -w qualifier always specifies the fallback formatted document page width to be used when the -r (override format) qualifier is specified or when the document has no inherent format. When used with the -p (page mode) qualifier, the display page width is determined from the terminal and is unrelated to the formatted page width. In nonpage mode, the specified -w value is used for both fallback document page width and the display page width. If the -w qualifier is not specified, the default width is 80 characters.

–h *paper_height* Specifies the paper height in units of characters. The -h qualifier always specifies the fallback formatted document page height to be used when the -r (override format) qualifier is specified or when the document has no inherent format. When used with the -p (page mode) qualifier, the display page height is determined from the terminal and is unrelated to the formatted page height. In nonpage mode, the specified -h value is used for both fallback document page height and the display page height. If the -h qualifier is not specified, the default height is dependent on the document.

–p Specifies that the CDA Viewer is to pause after displaying each page. The user can also page backward and go directly to the top or bottom of the document. If the -p qualifier is not specified, the CDA Viewer displays each page without pausing.

See Also

cdoc(1), dxvdoc(1X), DDIF(5), DTIF(5), CDA(5), DOTS(5)

Name

vi – screen editor

Syntax

vi [**–t** *tag*] [*+command*] [**–l**] [**–r**] [**–w***n*] [**–x**] *name...*

Description

The `vi` (visual) editor is a display-oriented text editor based on `ex`(1). The `ex` command and the `vi` command run the same code. You can access the command mode of `ex` from within `vi`.

The following is a list of some of the `vi` commands. See the *Vi Beginner's Reference* card and the *Introduction to Display Editing with Vi* for more details on using `vi`.

Screen Control Commands

<CTRL/L>	Reprints current screen.
<CTRL/Y>	Exposes one more line at top of screen.
<CTRL/E>	Exposes one more line at bottom of screen.

Paging Commands

<CTRL/F>	Pages forward one screen.
<CTRL/B>	Pages back one screen.
<CTRL/D>	Pages down half screen.
<CTRL/U>	Pages up half screen.

Cursor Positioning Commands

j	Moves cursor down one line, same column.
k	Moves cursor up one line, same column.
h	Moves cursor back one character.
l	Moves cursor forward one character.
<RETURN>	Moves cursor to beginning of next line.
0	Moves cursor to beginning of current line.
$	Moves cursor to end of current line.
<SPACE>	Moves cursor forward one character.
*n***G**	Moves cursor to beginning of line *n*. Default is last line of file.
/pattern	Moves cursor forward to next occurrence of *pattern*.
?*pattern*	Moves cursor backward to next occurrence of *pattern*.
n	Repeats last / or ? pattern search.

Text Insertion Commands

a	Appends text after cursor. Terminated by <ESC>.

A	Appends text at the end of the line. Terminated by <ESC>.
i	Inserts text before cursor. Terminated by <ESC>.
I	Inserts text at the beginning of the line. Terminated by <ESC>.
o	Opens new line below the current line for text insertion. Terminated by <ESC>.
O	Opens new line above the current line for text insertion. Terminated by <ESC>.
<DELETE>	Overwrites last character during text insertion.
<ESC>	Stops text insertion.

Text Deletion Commands

dw	Deletes current word.
x	Deletes current character.
dd	Deletes current line.
D, d$	Deletes from cursor to end of line.
P	Puts back text from the previous delete.

Text Change Commands

cw	Changes characters of current word until stopped with escape key.
c$	Changes text up to the end of the line.
C, c$	Changes remaining text on current line until stopped by pressing the escape key.
~	Changes case of current character.
xp	Transposes current and following characters.
J	Joins current line with next line.
r*x*	Replaces current character with *x*.

Buffer Usage Commands

[a–z]n yy	Yanks *n* lines to the *[a–z]* buffer. Default is current line.
[a–z]n p	Puts *n* yanked text lines from the *a–z* buffer, after the cursor.

Exiting vi

ZZ	Exits `vi` and saves changes
:wq	Writes changes to current file and quits edit session.
:q	Quits edit session (no changes made).

The `view` command uses all of the same edit or commands as `vi`. However, `view` does not allow you to write the file. See `view`(1).

Options

–t *tag* Specifies a list of *tag* files. The *tag* files are preceded by a backslash (\) and are separated by spaces. The *tag* option should always be the first entry.

+command Tells the editor to begin by executing the specified *command.* A useful example would be *+/pattern* to search for a *pattern.*

–l Sets the showmatch and lisp options for editing LISP code.

–r *name* Retrieves the last saved version of the *name*'d file in the event of an editor or system crash. If no file is specified, a list of saved files is produced.

–w*n* Sets the default window size to *n*. This option is useful for starting in a small window on dialups.

NOTE

The **–x** option is available only if the Encryption layered product is installed.

–x Causes `vi` to prompt for a *key*. The *key* is used to encrypt and decrypt the contents of the file. If the file contents have been encrypted with one *key*, you must use the same *key* to decrypt the file.

Restrictions

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.

The *wrapmargin* option sometimes works incorrectly because it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line is not broken.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give on a **:global** you must **Q** to `ex` command mode, execute them, and then reenter the screen editor with `vi` or `open`.

See Also

ed(1), ex(1), view(1)
"An Introduction to Display Editing with Vi", *ULTRIX Supplementary Documents, Vol. I: General User*

Name

view – displays a file using the vi commands

Syntax

view [**–t** *tag*] [**–r**] [**+***command*] [**–l**] [**–w***n*] [**–x**] *name...*

Description

The `view` command displays a text file. The `view` command and the `vi` command run almost the same code except that in `view` changes to a file are not allowed. It is possible to get to the command mode of `ex` from within both.

The following is a list of some of the `view` commands. See the *vi Beginner's Reference* card and the *Introduction to Display Editing with vi* for more details that can be helpful for using `view`.

Screen Control Commands

<CTRL/L>	Reprints current screen.
<CTRL/Y>	Exposes one more line at top of screen.
<CTRL/E>	Exposes one more line at bottom of screen.

Paging Commands

<CTRL/F>	Pages forward one screen.
<CTRL/B>	Pages back one screen.
<CTRL/D>	Pages down half screen.
<CTRL/U>	Pages up half screen.

Cursor Positioning Commands

j	Moves cursor down one line, same column.
k	Moves cursor up one line, same column.
h	Moves cursor back one character.
l	Moves cursor forward one character.
<RETURN>	Moves cursor to beginning of next line.
0	Moves cursor to beginning of current line.
$	Moves cursor to end of current line.
<SPACE>	Moves cursor forward one character.
*n***G**	Moves cursor to beginning of line *n*. Default is last line of file.
/pattern	Moves cursor forward to next occurrence of *pattern*.
?*pattern*	Moves cursor backward to next occurrence of *pattern*.
n	Repeats last / or ? pattern search.

Exiting view

ZZ	Exits `view`.
:q	Quits `view` session.

Options

–t *tag* Specifies a list of *tag* files. The *tag* files are preceded by a backslash (\) and are separated by spaces. The *tag* option should always be the first entry.

+*command* Tells the editor to begin by executing the specified *command.* An example would be +/*pattern* that would search for a *pattern.*

–l Sets the showmatch and lisp options for viewing LISP code..

–r Retrieves the last saved version of the *name*'d file in the event of a system crash. If no file is specified, a list of saved files is produced.

–w*n* Sets the default window size to *n*. This option is useful for starting in a small window on dialups.

NOTE

The **–x** option is available only if the Encryption layered product is installed.

–x Causes `view` to prompt for a *key*. The *key* is used to encrypt and decrypt the contents of the file. If the file has been encrypted with one *key*, you must use the same *key* to decrypt the file.

See Also

edit(1), ex(1), vi(1)
"An Introduction to Display Editing with vi" in the *ULTRIX Supplementary Documents* Vol. I: General User

vmstat (1)

Name

vmstat – report virtual memory statistics

Syntax

vmstat [*interval* [*count*]]
vmstat –v [*interval* [*count*]]
vmstat –fKSsz
vmstat –Kks *namelist* [*corefile*]

Description

The `vmstat` command reports statistics on processes, virtual memory, disk, trap, and cpu activity.

If `vmstat` is specified without arguments, this command summarizes the virtual memory activity since the system was last booted. If the *interval* argument is specified, then successive lines are summaries of activity over the last *interval* seconds. Because many statistics are sampled in the system every five seconds, five is a good specification for *interval;* other statistics vary every second. If the *count* argument is provided, the statistics are repeated *count* times.

When you run `vmstat` the format fields are as follows:

Procs: information about numbers of processes in various states.

- **r** in run queue
- **b** blocked for resources (i/o, paging, and so on.)
- **w** runnable or short sleeper (< 20 seconds) but swapped

faults: trap/interrupt rate averages per second over the last 5 seconds.

- **in** (non clock) device interrupts per second
- **sy** system calls per second
- **cs** cpu context switch rate (switches/second)

cpu: breakdown of percentage usage of cpu time

- **us** user time for normal and low priority processes
- **sy** system time
- **id** cpu idle time

Memory: information about the use of virtual and real memory. Virtual pages are considered active if they belong to processes which are running or have run in the last 20 seconds.

- **avm** active virtual pages
- **fre** size of the free list

Pages are reported in units of 1024 bytes.

If the number of pages exceeds 9999, it is shown in a scaled representation. The suffix k indicates multiplication by 1000 and the suffix m indicates multiplication by 1000000. For example, the value 12345 appears as 12k.

page: information about page faults and paging activity. These are averaged every five seconds, and given in units per second. The size of a unit is always 1024 bytes and is independent of the actual page size on a machine.

- **re** page reclaims (simulating reference bits)
- **at** pages attached (found in free list not swapdev or filesystem)
- **pi** pages paged in
- **po** pages paged out
- **fr** pages freed per second
- **de** anticipated short term memory shortfall
- **sr** pages scanned by clock algorithm, per-second

disk: s0, s1 ...sn: Paging/swapping disk sector transfers per second (this field is system dependent). Typically paging is split across several of the available drives. This will print for each paging/swapping device configured into the kernel.

Options

- **–f** Provides reports on the number of *forks* and *vforks* since system startup and the number of pages of virtual memory involved in each kind of fork.
- **–K** Displays usage statistics of the kernel memory allocator.
- **–k** Allows a dump to be interrogated to print the contents of the sum structure when specified with a *namelist* and *corefile*. This is the default.
- **–S** Replaces the page reclaim (re) and pages attached (at) fields with processes swapped in (si) and processes swapped out (so).
- **–s** Prints the contents of the *sum* structure, giving the total number of several kinds of paging related events that have occurred since boot.
- **–v** Prints an expanded form of the vitual memory statistics.
- **–z** Zeroes out the sum structure if the UID indicates root privilege.

Examples

The following command prints what the system is doing every five seconds:

```
vmstat 5
```

To find the status after a core dump use the following:

```
cd /usr/adm/crash
vmstat -k vmunix.? vmcore.?
```

Files

- `/dev/kmem` Kernel memory
- `/vmunix` System namelist

w(1)

Name

w – display who is logged in and what they are doing

Syntax

w [*options*] [*user*]

Description

The w command prints a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields output are:
The users login name
The name of the tty the user is on
The host from which the user is logged in
The time of day the user logged on
The number of minutes since the user last typed anything
The CPU time used by all processes and their children on that terminal
The CPU time used by the currently active processes
The name and arguments of the current process

Options

–d Outputs debug information.

–f Suppresses the 'from' field.

–h Suppresses the normal header from the output.

–l Displays information in long format (default).

–s Displays information in short format. In the short form, the tty is abbreviated, the login time and cpu times are left off, as are the arguments to commands.

–u Outputs the same information as the `uptime` command.

If a *user* name is included, the output will be restricted to that user.

Restrictions

The notion of the ''current process'' is unclear. The current algorithm is ''the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal''. This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. (In cases where no process can be found, w prints ''–''.)

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is ''charged'' with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

The w command does not know about conventions for detection of background jobs. It will sometimes find a background job instead of the right one.

Files

/etc/utmp
/dev/kmem
/dev/drum

See Also

finger(1), ps(1), who(1)

wait(1)

Name

wait – wait for process completion

Syntax

wait [*pid*]

Description

The `wait` command waits until all processes started with an ampersand (&) have completed and reports on abnormal terminations.

If a numeric *pid* is given and is the process ID of a background process, then `wait` waits until that process is completed. If *pid* is not a background process, `wait` waits until all background processes have completed.

Because the `wait`(2) system call must be executed in the parent process, the Shell itself executes `wait` without creating a new process.

Restrictions

Because not all the processes of a 3- or more-stage pipeline are children of the Shell, the `wait` command does not work on them.

The [*pid*] is available only with sh5.

See Also

sh(1), sh5(1), wait(2)

Name

wall – write to all users

Syntax

wall

Description

The `wall` command reads its standard input until an EOF. It then sends this message, preceded by 'Broadcast Message ...', to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

Diagnostics

'Cannot send to ...' when the open on a user's tty file fails.

Files

/dev/tty?
/etc/utmp

See Also

mesg(1), write(1)

wc(1)

Name

wc – count words, lines, and characters

Syntax

wc [–lwc] [*name...*]

Description

The wc command counts lines, words and characters in the named files, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs or new lines.

If an argument beginning with one of ''lwc'' is present, the specified counts (lines, words, or characters) are selected by the letters **l**, **w**, or **c**. The default is **–lwc**.

Options

–c Displays number of characters only.

–l Displays number of lines only.

–w Displays number of words only.

Name

what – display ID keywords from SCCS file

Syntax

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
```

Use `what` in conjunction with the SCCS command `get`(1), which automatically inserts identifying information, but information can also be inserted manually.

Restrictions

It is possible that an unintended occurrence of the pattern **@(#)** could be found. This causes no harm in nearly all cases.

Diagnostics

Use `sccshelp`(1) for explanations.

Options

-s Quit after finding the first occurrence of the pattern in the file.

See Also

get(1), help(1), sccs(1)
"An Introduction to the Source Code Control System," *ULTRIX Supplementary Documentation* Vol. II:Programmer

whatis (1)

Name

whatis – display command description

Syntax

whatis *command...*

Description

The `whatis` command looks up a given command and gives the header line from the manual section. You can then run the `man`(1) command to get more information. If the line starts ''name(section) ...'' you can do ''man section name'' to get the documentation for it. Try ''whatis ed'' and then you should do ''man 1 ed'' to get the manual.

The `whatis` command is actually just the **–f** option to the `man`(1) command.

Files

/usr/lib/whatis Data base

See Also

man(1), catman(8)

Name

whatnow – prompting front-end for send

Syntax

whatnow [–draftfolder *+folder*] **[–draftmessage** *msg*] **[–nodraftfolder] [–editor** *editorname*] **[–noedit] [–prompt** *string*] **[file] [–help]**

Description

After you have finished an editing session from `comp, dist, forw` or `repl;` the `whatnow` program prompts you for the next required action. Press <RETURN> at the `what now?` prompt, to see a list of the available options. These options are:

```
display [<switches>]
edit [<editorname> <switches>]
list [<switches>]
push [<switches>]
quit [-delete]
refile [<switches>] +folder
send [-watch <switches>]
whom [-check <switches>]
```

Use `display` if you have been using `repl` or `dist` and want to see the original message. Use `edit` [<editorname>switches] if you want to continue editing the draft. Use `list` to display the draft message. If you use `push, send` operates in the background and frees your terminal while the message is being sent.

Use `quit` to exit from `whatnow` and to save the draft message. The `quit -d` option will exit from `whatnow` and delete the draft message.

Use `refile`+folder to refile the draft message in a specified folder. The `send` option will cause the message to be delivered. Use `whom` to find out who will receive the mail when it is sent.

Unless the `-noedit` argument is set, the editor starts when `whatnow` is invoked.

For the `edit` response, any valid switch to the editor is valid. Similarly, for the `send` and `whom` responses, any valid switch to `send` and `whom` commands, respectively, are valid.

For the `push` response, use any valid switch to `send`. MH invokes `send` with the `-push` option.

For the `refile` response, any valid switch to the `fileproc` is valid.

For the `display` and `list` responses, any valid argument to the `lproc` is valid. If any non-switch arguments are present, then the pathname of the draft will be excluded from the argument list given to the `lproc` (this is useful for listing another MH message).

See `mh-profile`(5mh) for further information about how editors are used by MH. It also describes `proc` and `fileproc` and shows how complex variables can be used to direct `whatnow`'s actions.

Options

The default prompt is `What now?`. You can change this using the `-prompt` *string* option. You must encase the *string* in double quotes (" "), if you want the prompt string to include spaces. The following example would set the prompt to be "Now What?".

```
$ whatnow -prompt "Now What?"
```

The `whatnow` program normally searches for the draft message in your Mail directory. You can change this by using the `-draftfolder`<+folder> option. In the following example, `whatnow` would search in the +test folder for the draft message.

```
$ whatnow -draftfolder +test
```

This would be useful, only if you had previously created a draft message in a specific folder. See `comp`(1mh) for details.

You can also direct `whatnow` to search for a specific file by using the `draftmessage`<filename> option. If you do not specify a directory path, `whatnow` will assume that it is located in your Mail directory. Similarly, if you specify a directory pathname that is not absolute (that is, does not begin with a / ./ or a ../) `whatnow` will assume that the path is relative to your Mail directory and not your home directory.

If `sendproc` is `send`, then `whatnow` uses a built-in `send`, it does not actually run the `send` program. Hence, if you define your own `sendproc`, do not call it `send`, as `whatnow` will not run it.

Files

$HOME/.mh_profile	The user profile
<mh–dir>/draft	The draft file

Profile Components

Path:	To determine your MH directory
Draft-Folder:	To find the default draft-folder
Editor:	To override the default editor
<lasteditor>–next:	To name an editor to be used after exit from <lasteditor>
fileproc:	Program to refile the message
lproc:	Program to list the contents of a message
sendproc:	Program to use to send the message
whomproc:	Program to determine who a message would go to

See Also

comp(1mh), send(1mh), whom(1mh)

Name

whereis – locate source, binary, and or manual for program

Syntax

whereis [–sbm] [–u] [–SBM *dir...* **–f]** *name...*

Description

The `whereis` command locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form ".ext", for example,".c". Prefixes of "s." resulting from use of source code control are also dealt with. The `whereis` command then attempts to locate the desired program in a list of standard places.

Options

–S dir
Search for source files in specified directory.

–B dir
Search for binary files in given directory.

–M dir
Search for manual section files in given directory.

–b Searches only for binary files.

–f Terminates last directory list created from use of **–S**, **–B** or **–M** flags and signals the start of file names.

–m Searches only for manual section files.

–s Searches only for source files.

–u Searches for files that do not have one of binary, source or manual section files. A file is said to be unusual if it does not have one entry of each requested type. Thus "whereis -m -u *" asks for those files in the current directory which have no documentation.

Examples

The following finds all the files in `/usr/ucb` which are not documented in `/usr/man/man1` with source in `/usr/src/cmd`:

```
cd /usr/ucb
whereis -u -M /usr/man/man1 -S /usr/src/cmd -f *
```

Restrictions

Since the program uses `chdir`(2) to run faster, pathnames given with the **–M –S** and **–B** must be full. That is, they must begin with a "/".

Files

/usr/src/*
/usr/{doc,man}/*
/lib, /etc, /usr/{lib,bin,ucb,old,new,local}

Name

which – locate program file

Syntax

which [*name...*]

Description

The `which` command takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are taken from the user's .cshrc file.

Restrictions

Must be executed by a csh, since only csh's know about aliases.

Diagnostics

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

Files

~/.cshrc	source of aliases and path values

who(1)

Name

who – print who and where users are logged in

Syntax

who [*who-file*] [**am i**]

Description

The `who` command, without an argument, lists the login name, terminal name, and login time for each current UNIX user.

Without an argument, `who` examines the /etc/utmp file to obtain its information. If a file is given, that file is examined. Typically the given file will be /usr/adm/wtmp, which contains a record of all the logins since it was created. Then `who` lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with /dev/ suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with '~' in the place of the device name, and a fossil time indicative of when the system went down.

With two arguments, as in 'who am I' (and also 'who are you'), `who` tells who you are logged in as.

Files

/etc/utmp

See Also

getuid(2), utmp(5)

Name

whoami – print your current login name

Syntax

whoami

Description

The `whoami` command prints who you are. It works even if you are su'd, while 'who am i' does not since it uses `/etc/utmp`.

Files

/etc/passwd Name data base

See Also

who(1)

whom(1mh)

Name

whom – report to whom a message would go

Syntax

whom [–alias *aliasfile*] **[–check] [–nocheck] [–draft] [–draftfolder** *+folder*] **[–draftmessage** *msg*] **[–nodraftfolder] [file] [–help]**

Description

The whom command is used to expand the headers of a message into a set of addresses and optionally to verify that those addresses are deliverable at that time if -check is given.

Options

The whom program normally searches for the draft message in your Mail directory. You can change this by using the -draftfolder *+folder* option. In the following example, whom would search in the +test folder for the draft message.

```
% whom –draftfolder +test
```

This would be useful, only if you had previously created a draft message in a specific folder. See comp(1mh) for details.

You can also direct whom to search for a specific file by using the draftmessage *filename* option. If you do not specify a directory path, whom will assume that it is located in your Mail directory. Similarly, if you specify a directory pathname that is not absolute (does not begin with a \) whom will assume that the path is relative to your Mail directory and not your home directory.

By using the -alias aliasfile switch, you can direct send to consult the named files for alias definitions. You can reference more than one file, but each filename must be preceded by the word -alias. See mh-alias(5mh) for more information.

With the -check option, whom makes no guarantees that the addresses listed as being correct are really deliverable: rather, an address being listed as correct means that at the time that whom was run the address was thought to be deliverable by the transport service. For local addresses, this is absolute; for network addresses, it means that the host is known; for uucp addresses, it means that the UUCP network is available for use.

The defaults for this command are:

```
file defaults to <mh–dir>/draft
-nocheck
-alias /usr/new/lib/mh/MailAliases
```

Files

$HOME/.mh_profile The user profile

Profile Components

`Path:`	To determine your MH directory (mh-dir)
`Draft-Folder:`	To find the default draft-folder
`postproc:`	Program to post the message

See Also

comp(1mh), mh-alias(5mh), post(8mhs)

Name

write – write message to another user

Syntax

write *user* [*ttyname*]

Description

The `write` command copies lines from your terminal to that of another user. When first called, it sends the message

```
Message from yoursystem!yourname yourttyname...
```

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point `write` writes 'EOT' on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

Permission to write may be denied or granted by use of the *mesg* command. At the outset writing is allowed. Certain commands, in particular `nroff` and `pr`(1) disallow messages in order to prevent messy output.

If the character '!' is found at the beginning of a line, `write` calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using `write`: when you first write to another user, wait for him to write back before starting to send. Each party should end each message with a distinctive signal. The letter 'o' is the convention for 'over' which indicates that the message is complete. The letters 'oo' are the convention for 'over and out' which is used when the conversation is about to be terminated.

Files

/etc/utmp	to find user
/bin/sh	to execute '!'

See Also

mail(1), mesg(1), who(1)

Name

xargs – construct argument list and execute command

Syntax

xargs [*flags*] [*command* [*initial-arguments*]]

Description

The command `xargs` combines fixed *initial-arguments* with arguments read from standard input to execute a specified *command* one or more times. The number of arguments read when a *command* is invoked and how they are combined is determined by the options specified.

The specified *command,* (which can be a Shell file) is searched for using ones' **$PATH** specification. If *command* is not specified, **/bin/echo** is used.

Arguments read from standard input are defined as contiguous strings of characters delimited by one or more blanks, tabs, or newlines; empty lines are always discarded. Blanks and tabs can be embedded as part of an argument if they contain an escape character or if they are quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed; a backslash (\) escapes the next character.

Options

Each argument list begins with the *initial-arguments,* followed by arguments read from standard input, with the exception of the **-i** option. See the description of the **–i** option for more information.

The options **–i, –l,** and **–n** determine how arguments are selected when each command is invoked. If none of these options are specified, the *initial-arguments* are followed by arguments read continuously from standard input until the internal buffer is full; then, *command* executes with the accumulated arguments. This process repeats until no arguments exist. When conflicts arise, such as the **–l** option used with the **–n,** the last option has precedence. The options values are as follows:

–l*number*

Execute *command* for each non-empty *number* lines of arguments from standard input. When command is invoked for the final time, it has fewer lines of arguments if fewer than a specified *number* remain. A line ends with the first newline unless the last character of the line is a blank or a tab; a trailing blank or tab signals continuation through the next non-empty line. If *number* is is not specified, the value 1 is assumed. The option **–x** is forced.

–i*replstr* **(Insert mode)**

Execute *command* for each line from standard input, taking the entire line as a single argument and inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of five arguments specified in *initial-arguments* can contain one or more occurrence of *replstr*. Blanks and tabs at the beginning of each line are discarded. A constructed arguments cannot exceed 255 characters and the option **–x** is a forced. A {|} is assumed for *replstr* if not specified.

–n*number*

Execute *command* using as many standard input arguments as possible, up to the specified *number* arguments maximum. Fewer arguments are used if their total size is greater than *size* characters, and when the last command is invoked, fewer *number* of arguments remain. If the option **–x** is also include, each specified *number* of arguments must fit in the *size* limitation, or else *xargs* terminates execution.

–t (Trace mode)
Echo the *command* and each constructed argument list to file descriptor 2 prior to their execution.

–p (Prompt mode)
Asks the user whether or not *command* should be executed each time *command* is invoked. Trace mode (**–t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** executes the command; any other response does not invoke that particular *command*.

–x Causes the command *xargs* to terminate if an argument list is greater than the specified *size* of characters; the option **–x** is forced by the options **–i** and **–l**. When the options **–i**, **–l**, or **–n** are included, the total length of all arguments must be within the specified *size* limit.

–s*size* The maximum 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 included, 470 is the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

–e*eofstr*
The option *eofstr* is taken as the logical end-of-file string. Underscore (_) is assumed for the logical **EOF** string if **–e** is not specified. The value **–e** without *eofstr* specified turns off the logical **EOF** string capability; the underscore is taken literally. The command *xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

The command *xargs* terminates if it receives a return code of **–1** from *command* or if it cannot execute *command*. When *command* is a Shell program, it should explicitly *exit* with an appropriate value to avoid returning with **–1**. See `sh`(1) for more information.

Examples

The following example moves all files from directory $1 to directory $2 and echoes the move command prior to executing it:

```
ls $1 | xargs –i –t mv $1/{ } $2/{ }
```

The following example combines 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
```

In the next example, the user is prompted to specify which files in the current directory are to be archived. The first example archives the files one at a time; the second example archives groups of files:

```
ls | xargs –p –l ar r arch

ls | xargs –p –l | xargs ar r arch
```

The following example executes *diff*(1) with successive pairs of arguments originally typed as Shell arguments:

```
echo $* | xargs -n2 diff
```

See Also

sh(1).

xsend(1)

Name

xsend, xget, enroll – secret mail (available only if the Encryption layered product is installed)

Syntax

xsend *person*
xget
enroll

Description

This reference page describes software that is available only if the Encryption layered product is installed.

These commands implement a secure communication channel on a local machine. Mail sent using `xsend` is the same as `mail` but no one can read the message except the intended recipient.

To receive messages use, `enroll`. The `enroll` command asks for a password that you must enter to receive secret mail.

To receive secret mail use, `xget`. The `xget` command asks for your password, then gives you the messages.

To send secret mail, use `xsend` in the same manner as the ordinary mail command; however, `xsend` will accept only one target. A message announcing the receipt of secret mail is sent by ordinary mail.

Files

`/usr/spool/secretmail/*.key:`
keys

`/usr/spool/secretmail/*.[0-9]:`
messages

See Also

binmail(1), mail(1), crypt(1)

Name

xstr – extract strings from C program

Syntax

xstr [**–c**] [**–**] [*file*]

Description

The `xstr` command maintains a file *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

```
xstr -c name
```

will extract the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An appropriate declaration of `xstr` is prepended to the file. The resulting C text is placed in the file x.c, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file xs.c declaring the common `xstr` space can be created by a command of the form

```
xstr
```

This xs.c file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

The `xstr` command can also be used on a single file. A command

```
xstr name
```

creates files x.c and xs.c as before, without using or affecting any *strings* file in the same directory.

It may be useful to run `xstr` after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. The `xstr` command reads from its standard input when the argument '–' is given. An appropriate command sequence for running `xstr` after the C preprocessor is:

```
cc -E name.c | xstr -c -
cc -c x.c
mv x.o name.o
```

The `xstr` command does not touch the file *strings* unless new items are added, thus `make` can avoid remaking xs.o unless truly necessary.

Options

- Reads stdin.

-c Extracts strings from specified C source (next argument).

Restrictions

If a string is a suffix of another string in the data base, but the shorter string is seen first by `xstr` both strings will be placed in the data base, when just placing the longer one there will do.

Files

strings	Data base of strings
x.c	Massaged C source
xs.c	C source for definition of array 'xstr'
/tmp/xs*	Temp file when 'xstr name' doesn't touch *strings*

See Also

mkstr(1)

Name

yacc – yet another compiler-compiler

Syntax

yacc [–vd] *grammar*

Description

The `yacc` command converts a context-free grammar into a set of tables for a simple automaton which executes an left recursive 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`.

Options

–d Writes all define statements to `y.tab.h` file. This allows source files other than `y.tab.c` to access the token codes.

–v Writes description of parsing tables and report of grammatical conflicts to `y.output` file.

Diagnostics

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the `y.output`file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

Restrictions

Because file names are fixed, at most one `yacc` process can be active in a given directory at a time.

Files

y.output
y.tab.c
y.tab.h defines for token names
yacc.tmp, yacc.acts temporary files

See Also

lex(1)
"YACC – Yet Another Compiler Compiler" *ULTRIX Supplementary Documents* Vol. II:Programmer

yes (1)

Name

yes – be repetitively affirmative

Syntax

yes [*arg*]

Description

The `yes` command repeatedly outputs "y". If *arg* is given, that argument is output repeatedly. Terminate the `yes` command with <CTRL/C>

Name

ypcat – print values from a YP data base

Syntax

ypcat [–k] [–t] [–d *domainname*] *mname*
ypcat –x

Description

The `ypcat` command prints values stored in a yellow pages (YP) map specified by *mname,* which may be either a *mapname* or a map *nickname.*

To look at the network-wide password database, *passwd.byname,* with the nickname *passwd,* type:

```
ypcat passwd
```

Options

–d *domainname*
Displays information on the domain specified by *domainname.*

–k Displays keys for maps in which values are null or key is not part of the value.

–t Inhibits translation of *mname* to *mapname.* For example,

```
ypcat -t passwd
```

will fail because there is no map named *passwd,* whereas

```
ypcat passwd
```

will be translated to

```
ypcat passwd.byname.
```

–x Displays map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

See Also

domainname(1yp), ypmatch(1yp), ypfiles(5yp), ypserv(8yp)

ypmatch (1yp)

Name

ypmatch – print the value of one or more keys from a yp map

Syntax

ypmatch [**–d** *domain*] [**–k**] [**–t**] *key... mname*
ypmatch –x

Description

The `ypmatch` command prints the values associated with one or more keys from the yellow pages (YP) map (database) specified by a *mname,* which may be either a *mapname* or a map *nickname.*

Multiple keys can be specified. After the key values and the map name have been specified, `ypmatch` searches the map for all of the specified keys. The specified keys must be exact values in terms of capitalization and length. The `ypmatch` command does not have a pattern matching capability. If `ypmatch` cannot match a key, it produces a diagnostic message.

Options

–d Displays key values for specified domain.

–k Displays key, followed by a colon (:), before displaying value of the key. This is useful if the keys are not duplicated in the returned values, or if the number of specified keys is so large that the output is confusing.

–t Inhibits translation of nickname to mapname. For example,

```
ypmatch -t zippy passwd
```

fails because there is no map named *passwd,* while

```
ypmatch zippy passwd
```

succeeds because `ypmatch` translates it to

```
ypmatch zippy passwd.byname.
```

–x Displays map nickname table. This option tells `ypmatch` to list the nicknames (*mnames*) with their associated *mapnames.*

See Also

ypfiles(5yp), ypcat(1yp)

Name

yppasswd – change password in yellow pages (YP) service.

Syntax

yppasswd [*name*]

Description

The `yppasswd` command lets you change your password in the yellow pages (YP) map, a network data base service. Only you or the superuser can change your YP password.

When you enter the `yppasswd` command, the program prompts you for the old password and then for the new password. Note that the passwords are not displayed on the screen.

Next, the program asks you for the new password again, to verify that you have typed it correctly. If you do not type the passwords correctly, you will receive an error message after you enter the new password.

Your new YP password must meet **one** of the following requirements:

- It must be a combination of at least six alphanumeric characters, **or**
- It must be a minimum of four characters, with at least one being non-alphanumeric, such as a control sequence.

NOTE

The `passwd` command does not change the YP password. This command only changes the local password file (`/etc/passwd`), and not the YP master password file. See Chapter 3 of the *Guide to the Yellow Pages Service* for further information.

Diagnostics

Please use a longer password
Your new password does not meet the minimum length requirement.

Mismatch- password unchanged
You misspelled your new password or its verification.

couldn't change passwd
Your new password can not be activated. It must be different from your old password and your login name.

Files

`/etc/passwd`	Password file
`/etc/yp`	Yellow Pages directory

See Also

passwd(1yp), passwd(5yp), ypfiles(5yp), yppasswdd(8yp)
Guide to the Yellow Pages Service

Name

ypwhich – determine which host is the current YP server or map master.

Syntax

ypwhich [**–d** *domain*] [**–V1**] [**–V2**] [*hostname*]
ypwhich [**–d** *domain*] [**–m** *mname*] [**–t**]]
ypwhich –x

Description

The `ypwhich` command identifies the YP server that currently supplies yellow pages services to a YP client. It also identifies which YP server is the master for a map. If invoked without arguments, `ypwhich` returns the host name of the YP server for the local machine. If *hostname* is specified, `ypwhich` checks that machine to find out which YP master it is using.

Refer to `ypfiles`(5yp) and `ypserv`(8yp) for an overview of the yellow pages.

Options

–V1
Identifies which server is serving v.1 YP protocol-speaking client processes.

–V2
Identifies which server is serving v.2 YP protocol-speaking client processes.

If neither version is specified, `ypwhich` attempts to locate the server that supplies the current v.2 services. If there is no v.2 server currently bound, `ypwhich` attempts to locate the server supplying the v.1 services. Since YP servers and YP clients are both backward compatible, the user need seldom be concerned about which version is currently in use.

–d Uses *domain* instead of the current domain.

–m *mname*
Finds the master YP server for a map. No *hostname* can be specified with **–m.** The *mname* argument can be a mapname, or a nickname for a map.

–t Inhibits nickname translation and is useful if there is a mapname identical to a nickname.

–x Displays the map nickname table. This option lists the nicknames (*mnames*) that the command knows of, and indicates the *mapname* associated with each nickname.

See Also

ypfiles(5yp), rpcinfo(8nfs), ypserv(8yp), ypsetup(8yp)

Index

Special Characters

Numbers

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-baud modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal*	————	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

What do you like best about this manual? _____

What do you like least about this manual? _____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear – Fold Here and Tape

digital™

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST–CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062–9987

Do Not Tear – Fold Here

Cut
Along
Dotted
Line

Reader's Comments

ULTRIX
Reference Pages Section 1: Commands M - Z
AD-PC0WA-T1

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

What do you like best about this manual? _____

What do you like least about this manual? _____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear – Fold Here and Tape

digital™

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST–CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062–9987

Do Not Tear – Fold Here

Cut
Along
Dotted
Line