NAME

intro - introduction to general commands (tools and utilities)

DESCRIPTION

The manual pages in section one contain most of the commands which comprise the user environment. Some of the commands included in section one are text editors, command shell interpreters, searching and sorting tools, file manipulation commands, system status commands, remote file copy commands, mail commands, compilers and compiler tools, formatted output tools, and line printer commands.

All commands set a status value upon exit which may be tested to see if the command completed normally. The exit values and their meanings are explained in the individual manuals. Traditionally, the value 0 signifies successful completion of the command.

SEE ALSO

man(1), intro(2), intro(3), intro(4), intro(5), intro(6), intro(7), intro(8), intro(9)

HISTORY

An **intro** manual appeared in Version 6 AT&T UNIX.

NAME

man - display manual pages

SYNOPSIS

man [**-k**] [*section*] *name* ...

DESCRIPTION

The **man** utility displays the manual pages entitled *name*. Pages may be selected according to a specific category (*section*).

The options are as follows:

-k A synonym for apropos(1). Instead of *name*, an expression can be provided using the syntax described in the apropos(1) manual. By default, it displays the header lines of all matching pages.

section Restricts the directories that **man** will search to a specific section. The currently available sections are:

1	General commands (tools and utilities).
2	System calls and error numbers.
3	Libraries.
4	Device drivers.
5	File formats.
6	Games.
7	Miscellaneous.
8	System maintenance and operation commands.
9	Kernel internals.
local	Manual pages located in */usr/local*.

The **man** configuration file, man.conf(5), specifies the possible *section* values, and their search order. Additional sections may be specified.

Guidelines for writing manual pages can be found in mdoc(7).

FILES

/etc/man.conf default man configuration file

EXIT STATUS

The **man** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

Display this manual page:

```
$ man man
```

Display the printf(3) manual page:

```
$ man 3 printf
```

List all manual pages with ‘intro’ in the header line:

```
$ man -k intro
```

SEE ALSO

apropos(1), intro(1), whatis(1), whereis(1), intro(2), intro(3), intro(4), intro(5), man.conf(5), intro(6), intro(7), mdoc(7), intro(8), intro(9)

HISTORY

A **man** command appeared in Version 3 AT&T UNIX.

NAME

sh - command language interpreter

SYNOPSIS

sh [**-bCfn**] [**-c** *string* | **-s** | *file*]

DESCRIPTION

The **sh** utility is a *command language interpreter*: it reads its input, breaks it down into parts, and then executes those parts. Its chief uses are in interfacing between the user and the operating system, reading commands on the command line, and in chaining together groups of commands in a very flexible manner, through a shell script.

The shell receives input as follows:

-c *string*	Read commands from *string*.
-s	Read commands from standard input (the default).
file	Read commands from *file*.

The options below can be specified with a '**+**' rather than '**-**', meaning to unset the option. They can also be set or unset using the **set** command.

-b notify. The user is given notice asynchronously when background jobs complete.

-C noclobber. Do not permit the redirection operator ('>') to clobber (overwrite) existing files.

-f noglob. Do not expand file name patterns.

-n noexec. Read commands but do not execute them - useful for checking syntax errors in scripts. This option is ignored for interactive shells.

BUILTINS

The shell has a number of *built-ins* available: utilities that are included as part of the shell. The shell does not need to search for them and can execute them directly.

A number of built-ins are special in that a syntax error can cause a running shell to abort, and, after the built-in completes, variable assignments remain in the current environment. The following built-ins are special: **.**, **:**, **break**, **continue**, **eval**, **exec**, **exit**, **export**, **readonly**, **return**, **set**, **shift**, **times**, **trap**, and **unset**.

The built-ins available to **sh** are listed below. Unless otherwise indicated, they exit 0 on success, and >0 if an error occurs.

. *file*
Execute the commands in *file*, in the current environment. The actual file need not be executable, and its location is determined by searching PATH if there are no slashes in the filename. The exit status is that of the last command returned, or zero if no commands were executed. If no readable file can be found, a non-interactive shell will abort; an interactive shell writes an error message and returns a non-zero exit status.

: [*arg ...*]
The **:** command does nothing - it is a placeholder for when a command is required. Its exit status is always zero.

alias [*name*[=*value*] ...]
Define an alias *name* to *value*; when the shell encounters a command name that is an alias, its value is substituted. If *value* ends in a blank, the next word is checked for alias substitution too. If only a *name* is specified, display the value of that alias; if no arguments are given, list all aliases and their values. Aliases are visible in the current environment and that of subshells, but not by the parent process of the current shell or by utilities invoked by it.

bg [*id* ...]
Select a job by *id* (see the **jobs** command, below) to run in the background. The default job is "%+".

break [*n*]
Exit from the innermost **for**, **while**, or **until** loop, or from loop level *n*.

cd [*dir*]
Change the current working directory to *dir*, or $HOME by default. If *dir* is set to '-', change to the previous working directory and print the (now current) working directory.

command [**-p** | **-V** | **-v**] *command* [*arg* ...]
Invoke *command* (and any optional arguments), overriding any functions with the same name, and without any of the properties that special built-ins have.

The options to **command** are as follows:

-p Use a default value for PATH to search for the command.
-V Do not invoke *command*, but identify how the shell will interpret it (such as a function or special built-in).

-v Do not invoke *command*, but identify the pathname the shell will use to run it.

The exit status is that of *command*, or 126 if *command* could not be invoked, or 127 if an error occurred in **command** itself or *command* could not be found. If the options **-V** or **-v** are given, the exit status is 0 on success, or >0 if an error occurs.

continue [*n*]
Go directly to the next iteration of the innermost **for**, **while**, or **until** loop, or from loop level *n*.

eval [*arg ...*]
Concatenate the arguments given and interpret them as a command. The exit status is that of the resulting command, zero if no arguments are given, or >0 if the resulting command could not be correctly parsed.

exec [*command* [*arg ...*]]
Replace the shell with *command* (and any optional arguments), without creating a new process. The exit status is that of *command*, or 126 if *command* could not be invoked, or 127 if *command* could not be found. If no command is given but a redirection happens, the exit status is 1-125; otherwise **exec** returns 0.

exit [*n*]
Exit the shell with exit status *n*, or that of the last command executed.

export [**-p**] *name*[=*value*] ...
Make the variable *name* visible to subsequently run commands, optionally setting it to *value*.

The options to the **export** command are as follows:

-p List all exported variables in a manner that can be reinput to the shell.

false
Return a false (non-zero) value.

fg [*id ...*]
Select a job by *id* (see the **jobs** command, below) to run in the foreground. The default job is "%+".

getopts *optstring name* [*arg ...*]
When invoked, **getopts** processes the positional parameters (or any *arg* passed to it) as a list of options and option arguments. **getopts** sets the variable *name* to the option found, OPTARG to its argument, and OPTIND to the index of the next variable to be processed.

The string *optstring* contains a list of acceptable options; a colon following an option indicates it may take an argument. If an option not recognised by *optstring* is found, *name* is set to '?'; if the first character of *optstring* is a colon, OPTARG is set to the unsupported option, otherwise an error message is displayed.

jobs [**-l** | **-p**] [*id ...*]
Display the status of all jobs in the current shell environment, or those selected by *id*.

The options to the **jobs** command are as follows:

-l Additionally display the process group ID.

-p Display only the process group ID.

Job *id* can be selected in one of the following ways:

%%	The current job.
%+	The current job.
%-	The previous job.
%n	Job number *n*.
%string	Job with command matching *string*.
%?string	Job with command containing *string*.

kill [**-l** [*signal*]] [**-s** *signal*] [**-***signal*] *pid ...*
Send a signal, by default SIGTERM, to the process with ID *pid*.

The options to the **kill** command are as follows:

-l [*signal*]	List all supported signals, or the signal name corresponding to *signal* number or the exit status of a command killed by a signal.
-s *signal*	Send the process *signal* name.
-*signal*	Send the process *signal* name or number.
pid	A process ID, process group ID, or a job ID (see **jobs**, above). The process ID 0 signals all processes in the current process group.

The supported signal numbers are:

0	Do not signal a process, but determine whether an ID exists.
1	SIGHUP: Terminal line hangup.
2	SIGINT: Interrupt a program.

3 SIGQUIT: Quit a program.
6 SIGABRT: Call abort(3).
9 SIGKILL: Kill a program. Cannot be caught or ignored.
14 SIGALRM: Real-time timer expired.
15 SIGTERM: Software termination signal.

pwd
Print the current working directory.

read [**-r**] *name ...*
Read a line from standard input. The line is split into fields, with each field assigned to a variable, *name*, in turn (first field assigned to first variable, and so on). If there are more fields than variables, the last variable will contain all the remaining fields. If there are more variables than fields, the remaining variables are set to empty strings. A backslash in the input line causes the shell to prompt for further input.

The options to the **read** command are as follows:

-r Ignore backslash sequences.

readonly [**-p**] *name*[=*value*]
Mark variable *name* as readonly, and optionally set it to *value*. Readonly variables cannot be later assigned values or unset.

The options to the **readonly** command are as follows:

-p Display the names and values of all readonly variables in a manner which can be reinput to the shell.

return [*n*]
Exit the current function or **.** script with exit status *n*, or that of the last command executed.

set [**-abCefhmnuvx**] [**-o** [*option*]] [*arg ...*]
Set options and positional parameters. Without options or arguments, display the names and values of all shell variables.

The options are described in the options description at the beginning of this manual. The sequence "set -o" displays the current option settings; the sequence "set +o" displays, in a format suitable to be reinput to the shell, a command suitable to achieve the current option settings.

Any arguments are assigned to the positional parameters, with the variable '#' set to the number of positional parameters. The sequence "set --" indicates an end to option processing (i.e. only arguments follow); "set --" by itself unsets all positional parameters and sets '#' to zero.

shift [*n*]
Shift the positional parameters *n* times (by default once). Parameter 1 takes the value of parameter '1+*n*', parameter 2 takes '2+*n*', and so on. Parameters '#' to '(#-*n*)+1' and downwards are unset and '#' is updated to the new number of positional parameters. If *n* is 0, no change occurs.

true
Return a true (zero) value.

umask [**-S**] [*mask*]

Set the file mode creation mask to *mask*. The creation mask determines the default permissions a newly created file will have. If *mask* is not specified, display the current creation mask.

The options to the **umask** command are as follows:

-S Display symbolic output.

See chmod(1) for the format of *mask*.

unalias [**-a**] *name ...*
Remove the alias definition of alias *name*.

The options to the **unalias** command are as follows:

-a Remove all alias definitions.

unset [**-fv**] *name ...*
Unset variable or function *name*.

The options to the **unset** command are as follows:

-f Treat *name* as a function.
-v Treat *name* as a variable (the default).

wait [*pid ...*]
Wait until all the processes specified by process or job ID *pid* have terminated. If no *pid* is specified, wait until all processes have terminated. The exit status is 0 on success, 1-126 if an error occurs, or 127 if *pid* was unknown.

SHELL GRAMMAR

The shell reads its input as described above. After that it follows a fairly simple chain of operations to parse that input:

- The shell breaks the input into *words* and *operators*. Words are the command text the user wishes run; operators are special characters which describe how the shell should interact with the commands.

- The shell *expands* the command text according to the rules of expansion.

- Words are subject to *field splitting*, where the command text is separated into commands and arguments to commands.

- The shell performs any *redirection*.

- The shell runs the commands. Argument names are assigned to *positional parameters*, with the command name itself assigned parameter 0.

- If the command is not being run in the background, the shell waits for it to complete and collects its exit status.

Quoting

Some characters have special meaning to the shell and need *quoting* if the user wants to indicate to the shell not to interpret them as such. The following characters need quoting if their literal meaning is desired:

| & ; < > ( ) $ ‘ \ " ’ <space> <tab> <newline>
* ? [# ~ = %

A backslash (\) can be used to quote any character except a newline. If a newline follows a backslash the shell removes them both, effectively making the following line part of the current one.

A group of characters can be enclosed within single quotes (') to quote every character within the quotes.

A group of characters can be enclosed within double quotes (") to quote every character within the quotes except a backquote (‘) or a dollar sign ($), both of which retain their special meaning. A backslash (\) within double quotes retains its special meaning, but only when followed by a backquote, dollar sign, double quote, or another backslash. An at sign (@) within double quotes has a special meaning (see *SPECIAL PARAMETERS*, below).

Similarly command words need to be quoted if they are not to be interpreted as such.

Expansion

Shell *variables* are arbitrary names assigned values using the ‘=’ operator; the values can be retrieved using the syntax $*variable*. Shell *parameters* are variable names, numbers, or any of the characters listed in *SPECIAL PARAMETERS*.

The shell is able to *expand* certain elements of its syntax, allowing for a more concise notation and providing a convenience to the user.

Firstly, tilde expansion occurs on words beginning with the ‘~’ character. Any characters following the tilde, up to the next colon or slash, if any, are taken as a login name and substituted with that user’s home directory, as defined in passwd(5). A tilde by itself is expanded

to the contents of the variable HOME. This notation can be used in variable assignments, in the assignment half, immediately after the equals sign or a colon, up to the next slash or colon, if any.

PATH=~alice:~bob/jobs

Parameter expansion happens after tildes have been expanded, with the value of the parameter being substituted. The basic format is:

${*parameter*}

The braces are optional except for positional parameters 10 and higher, or where the parameter name is followed by other characters that would prevent it from being expanded. If parameter expansion occurs within double quotes, neither pathname expansion nor field splitting happens afterwards.

Some special forms of parameter expansion are available. In the formats below, *word* itself is subject to expansion, and, if omitted, the empty string is used. If the colon is omitted, *word* is substituted only if *parameter* is unset (not if it is null).

${*parameter*:-[*word*]}
Substitute *parameter*. If *parameter* is unset or null, substitute *word*.

${*parameter*:=[*word*]}
Substitute *parameter*. If *parameter* is unset or null, first assign the value of *word* to *parameter*.

${*parameter*:?[*word*]}

Substitute *parameter*. If *parameter* is unset or null, the result of the expansion of *word* is written to standard error and the shell exits with a non-zero exit status. If *word* is omitted, the string "parameter null or not set" is used.

${*parameter*:+[*word*]}
Substitute *word*. If *parameter* is unset or null, substitute null.

${#*parameter*}
The length, in characters, of *parameter*.

${*parameter*%[*word*]}
Substitute *parameter*, deleting the smallest possible suffix matching *word*.

${*parameter*%%[*word*]}
Substitute *parameter*, deleting the largest possible suffix matching *word*.

${*parameter*#[*word*]}
Substitute *parameter*, deleting the smallest possible prefix matching *word*.

${*parameter*##[*word*]}
Substitute *parameter*, deleting the largest possible prefix matching *word*.

Command expansion has a command executed in a subshell and the results output in its place. The basic format is:

$(*command*)

or

‘*command*‘

The results are subject to field splitting and pathname expansion; no other form of expansion happens. If *command* is contained within double quotes, field splitting does not happen either. Within backquotes, a backslash is treated literally unless it follows a dollar sign, backquote, or another backslash. Commands can be nested, though the backquoted version requires backslashes before the backquotes. If *command* is run in a subshell in the bracketed version, the syntax is identical to that of arithmetic expansion. In that case the shell attempts arithmetic expansion first, then attempts command substitution if that fails. Or a non-ambiguous version can be used:

$((*command*))

Arithmetic expansion works similarly, with an arithmetic expression being evaluated and substituted. The format is:

$((*expression*))

Where *expression* is an integer, parameter name, or array reference, optionally combined with any of the operators described below, listed and grouped according to precedence:

() Operators within brackets have highest precedence. Compare 3+2*4, which is 11, since multiplication has higher precedence than addition, and (3+2)*4, which is 20.

+ - ~ ! Unary plus (indicates a positive value; integers are positive by default), unary minus (indicates a negative value), bitwise

NOT, and logical NOT (the result is 1 if the argument is zero, or 0 otherwise), respectively.

* / % Multiplication, division, and modulus (remainder), respectively.

+ - Addition and subtraction, respectively.

<< >> Shift left or right, respectively.

< <= > >=
Less than, less than or equal to, greater than, and greater than or equal to, respectively. The result is 1 if true, or 0 otherwise.

== != Equal (the result is 1 if both arguments are equal, and 0 otherwise) and not equal (the result is 1 if both arguments are non-zero, and 0 otherwise), respectively.

& Bitwise AND.

^ Bitwise exclusive OR.

| Bitwise inclusive OR.

&& Logical AND. The result is 1 if both arguments are non-zero, or 0 otherwise.

|| Logical OR. The result is 1 if either argument is non-zero, or 0 otherwise.

expression?*expr1*:*expr2*
The result is *expr1* if *expression* is non-zero, or *expr2* otherwise.

= *= /= %= += -= <<= >>= &= ^= |=
Assignment. The notation *var*=*expression* is equivalent to *var*=*var***expression*.

After the various types of expansion listed above have been carried out, the shell subjects everything that did not occur in double quotes to *field splitting*, where words are broken up according to the value of the IFS variable. Each character of IFS is used to split fields; any IFS characters at the beginning and end of input are ignored. If the value of IFS is null, no field splitting is performed.

After field splitting, the shell matches filename patterns.

? A question mark matches any single character.

* An asterisk matches multiple characters.

[..] Matches any character enclosed in the brackets. The sense is negated if the first character is '!'. A closing bracket can be included in the list of characters to match by listing it as the first character after the opening bracket or by quoting it. Similarly a '-' should be specified last or quoted so that the shell does not think it is a character range (see below).

[[:*class*:]]
Matches any character in the following character classes:

alnum	alpha	blank	cntrl
digit	graph	lower	print
punct	space	upper	xdigit

[*x*-*y*] Matches any character in the range between *x* and *y*, inclusive.

Slashes and full stops do not match the patterns above because of their use as path and filename characters.

Redirection

Redirection is used to open, close, or otherwise manipulate files, using redirection operators in combination with numerical *file descriptors*. A minimum of ten (0-9) descriptors are supported; by convention standard input is file descriptor 0, standard output file descriptor 1, and standard error file descriptor 2. In the examples given below, *n* represents a numerical file descriptor. The target for redirection is *file* and it is subject to all forms of expansion as listed above, except pathname expansion. If any part of the file descriptor or redirection operator is quoted, they are not recognised.

[*n*]<*file*

Open *file* for reading on file descriptor *n*, by default standard input.

[*n*]>*file*

Write to *file* with file descriptor *n*, by default standard output. If *file* does not exist, create it; if it does exist, truncate it to be empty before beginning to write to it.

[*n*]>|*file*

As above, but forces clobbering (see the **-C** option).

[*n*]>>*file*

Append to *file* with file descriptor *n*, by default standard output. If *file* does not exist, create it.

[*n*]<< This form of redirection, called a *here document*, is used to copy a block of lines to a temporary file until a line matching *delimiter* is read. When the command is executed, standard input is redirected from the temporary file to file descriptor *n*, or standard input by default. The basic format is:

[*n*]<<*delimiter*
text
text
...
delimiter

Provided *delimiter* doesn't contain any quoted characters, parameter, command, and arithmetic expansions are performed on the text block, and backslashes escape the special meaning of '$', '`', and '\'. If multiple here documents are used on the same command line, they are saved and processed in order.

[*n*]<<- Same as **<<**, except leading tabs are stripped from lines in *block*.

[*n*]<&*file*

Make file descriptor *n*, by default standard input, a copy of the file descriptor denoted by *file*. If *file* is '-', close file

descriptor *n* or standard input.

[*n*]>&*file*

Make file descriptor *n*, by default standard output, a copy of the file descriptor denoted by *file*. If *file* is '-', close file descriptor *n* or standard output.

[*n*]<>*file*

Open *file* for reading and writing on file descriptor *n*, by default standard input. If *file* does not exist, create it.

COMMANDS

The shell first expands any words that are not variable assignments or redirections, with the first field being the command name and any successive fields arguments to that command. It sets up redirections, if any, and then expands variable assignments, if any. It then attempts to run the command.

Firstly, if the command is a special built-in it invokes the built-in. If not, but it is a shell function, it then invokes that. If not, but it is a regular built-in, it then invokes that. Failing that, it uses the value of PATH to search for the command. If it finds a match which is a regular built-in or function it invokes it. Otherwise if it finds a match, or if the command name contains a slash, it attempts to execute the command in an environment separate from the shell. If it is unable to execute the command, it tries to run it as a shell script.

A series of commands separated by ';' constitute a *sequential list*, where commands are executed in the order given. The exit status of a sequential list is that of the last command executed. The format for a sequential list is:

command; *command* [; ...]

A series of commands separated by '&' constitute an *asynchronous list*, where the shell executes the command in a subshell and runs the next command without waiting for the previous one to finish. The exit status of an asynchronous list is always zero. The format for an asynchronous list is:

command & *command* [& ...]

A series of commands separated by '|' constitute a *pipeline*, where the output of one command is used as input for the next command. The exit status of a pipeline is that of the last command; if a pipeline begins '!' the exit status is inverted. The format for a pipeline is:

[!] *command* | *command* [| ...]

A series of commands separated by '&&' constitute an *AND list*, where a command is only executed if the exit status of the previous command was zero. The exit status of an AND list is that of the last command. The format for an AND list is:

command && *command* [&& ...]

A series of commands separated by '||' constitute an *OR list*, where a command is only executed if the exit status of the previous command was non-zero. The exit status of an OR list is that of the last command. The format for an OR list is:

command || *command* [|| ...]

A series of commands separated by '&&' and '||' constitute an *AND-OR list*, where '&&' and '||' have equal precedence and are evaluated in the order they are given. The AND-OR list can be terminated with ';' or '&' to have them execute sequentially or asynchronously, respectively.

Command lists, as described above, can be enclosed within '()' to have them executed in a subshell, or within '{}' to have them executed in the current environment:

(*command ...*)
{ *command ...*;}

Any redirections specified after the closing bracket apply to all commands within the brackets. An operator such as ';' or a newline are needed to terminate a command list within curly braces.

The shell has grammatical constructs which allow it to work its way (loop) through lists or evaluate things conditionally.

A *for loop* executes a series of commands for each item in a list. Its format is:

for *name* in *word ...*
do
  command
  ...
done

Firstly *word ...* is expanded to generate a list of items. The variable *name* is set to each item, in turn, and the commands are executed for

each item. The construct "in word ..." can be omitted, which is equivalent to: in "$@". The exit status is zero if there are no items or otherwise the exit status of the last command executed.

A *while loop* continuously executes a set of commands as long as the command has a zero exit status. Its format is:

```
while command
do
   command
   ...
done
```

Multiple commands may be given by grouping them in lists, as described above, or by separating them with newlines. The exit status is zero if the commands after "do" were never executed or otherwise the exit status of the last command executed.

An *until loop* continuously executes a set of commands as long as the command has a non-zero exit status. Its format is:

```
until command
do
   command
   ...
done
```

Multiple commands may be given by grouping them in lists, as described above, or by separating them with newlines. The exit status is zero if the commands after "do" were never executed or otherwise the exit status is that of the last command executed.

A *case conditional* is used to run commands whenever a pattern is matched. Its format is:

case *word* in
(*pattern* [| *pattern* ...]) *command*;;
...
esac

In this case *pattern* is matched against the string resulting from the expansion of *word*. Multiple commands may be given by grouping them in lists, as described above, or by separating them with newlines. The initial '(' is optional, as is the terminating ';;' for the final command. The exit status is zero if no patterns are matched or otherwise the exit status of the last command executed.

An *if conditional* is used to execute commands depending on the exit status of other commands. Its format is:

if *command*
then
command

elif *command*
then
command
...

else *command*
fi

Firstly the *command* following "if" is executed; if its exit status is

zero, the commands in the "then" block are executed and the conditional completes. Otherwise the commands in the "elif" block are executed; if the exit status is zero, the commands in the "then" block are executed and the conditional completes. Otherwise the next "elif" block, if any, is tried. If nothing from an "if" or "elif" block returns zero, the commands in the "else" block are run and the conditional completes. The "elif" and "else" blocks are optional. Multiple commands may be given by grouping them in lists, as described above, or by separating them with newlines. The exit status is zero if nothing is executed from an "if" or "elif" block or otherwise the exit status of the last command executed.

Functions allow the user to define a group of commands, executed whenever the function name is invoked. Its format is:

function() *command-list*

The above simply defines a function; nothing is executed until the function name is invoked. Commands may specify redirections and positional parameters are changed, for the duration of the function, to those passed to it. The special parameter '#' is temporarily changed too, though '0' is not. After the function finishes, the positional parameters and '#' are restored to their original values. The exit status of a function definition is 0 if successful or >0 otherwise. The exit status of a function is that of the last command executed by the function.

SPECIAL PARAMETERS

Some parameters have special meaning to the shell and are listed below.

0 The name of the shell or shell script.

1 ... n The *positional parameters*. These parameters are set when a shell, shell script, or shell function is invoked. Each argument passed to a shell or shell script is assigned a positional parameter, starting at 1, and assigned sequentially. When a shell function is invoked, any arguments passed to it are temporarily reassigned to the positional parameters; when the function completes, the values are restored. Positional parameters 10 and above should be enclosed in {}. Positional parameters can be reassigned using the **set** command.

@ All positional parameters. Within double quotes, each parameter is output as a separate field. The resulting list completely matches what was passed to the shell. So "1 2" "3" is output as two parameters, "1 2" and "3".

* All positional parameters. Within double quotes, all parameters are output as one field, separated by the first character of IFS (by default a space). The resulting list of words is amalgamated, losing the sense of how they were passed to the shell. So "1 2" "3" is output as one parameter, "1 2 3".

The number of positional parameters.

? The exit status of the most recent pipeline.

- The current shell options.

$ The process ID of the current shell. Subshells have the same

PID as the current shell.

! The process ID of the most recent background command.

ENVIRONMENT

The following environment variables affect the execution of **sh**:

CDPATH — Colon separated list of directories used by the **cd** command.

ENV — Pathname to a file containing commands to be executed when an interactive shell is started.

HISTFILE — Pathname to a file to be used to record command history. The default is to not write command history to a file.

HISTSIZE — The maximum number of commands stored in history. The default is 500.

HOME — Pathname to a user's home directory.

IFS — A list of characters to be used for field splitting.

LINENO — The current line number in a script or function, starting at 1. This variable should not be set by users.

MAIL — Pathname to a user's mailbox file. If set, **sh** reports the arrival of new mail (ascertained by checking a file's modification time) every

MAILCHECK seconds. MAIL is overridden by MAILPATH.

MAILCHECK — How often, in seconds, to check for new mail in either MAIL or MAILPATH. The default is 600 (10 minutes). If set to 0, check before issuing each prompt.

MAILPATH — Pathname to a colon separated list of mailboxes. If set, **sh** reports the arrival of new mail (ascertained by checking a file's modification time) every MAILCHECK seconds. The default notification message ("you have mail in $_") can be changed per mailbox by appending *%message* to a pathname. MAILPATH overrides MAIL.

OLDPWD — Pathname to the previous working directory.

OPTARG — An option argument for the **getopts** command.

OPTIND — An index to the next option for the **getopts** command.

PATH — Pathname to a colon separated list of directories used to search for the location of executable files. A pathname of '.' represents the current working directory. The default value of PATH on OpenBSD is:

/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11R6/bin:/usr/local/bin

PPID — The shell's parent process ID. Subshells have the same PPID as the parent of the current shell.

PS1 — User prompt displayed every time an interactive shell is ready to read a command. A '!' in the prompt is expanded to the number of the next command in history to be typed. The default value is '$ ' for normal users and '# ' for root.

PS2 — Newline prompt displayed in an interactive shell when a newline has been entered before the command line completes. The default value is '> '.

PS4 — Trace prompt displayed in an interactive shell before each command is traced (see the **-x** option). The default is '+ '.

PWD — The absolute pathname to the current working directory. Assignments to this variable are ignored.

ASYNCHRONOUS EVENTS

The following signals affect the execution of **sh**:

SIGINT — If a shell is interactive and in command line editing mode, editing is terminated on the current line and the command being edited is not entered into command history. Otherwise the signal is caught but no action is taken.

SIGQUIT Ignored if a shell is interactive.

SIGTERM Ignored if a shell is interactive.

SIGTSTP Ignored if a shell is interactive and the **monitor** option (**-m**) is set.

SIGTTIN Ignored if a shell is interactive and the **monitor** option (**-m**) is set.

SIGTTOU Ignored if a shell is interactive and the **monitor** option (**-m**) is set.

EXIT STATUS

The **sh** utility exits with one of:

0 The script being executed contained only blank lines or comments.

1-125 A non-interactive shell detected an error other than *file* not found.

126 A command was found but was not executable.

127 A non-interactive shell returned *file* not found.

Otherwise **sh** returns the exit status of the last command it invoked.

SEE ALSO

csh(1), ed(1), ksh(1), vi(1), script(7)

NAME

at, **batch** - queue, examine or delete jobs for later execution

SYNOPSIS

at [**-f** *file*] [**-l**] **-t** *time_arg*
at -r *job ...*
batch [**-f** *file*]

DESCRIPTION

at and **batch** read commands from standard input or a specified file which are to be executed at a later time.

at Executes commands at a specified time.

batch Executes commands when system load levels permit.

The options are as follows:

-f *file* Read the job from *file* rather than standard input.

-l Display the queue of jobs which are currently awaiting execution.

-r *job ...*
Remove the specified job(s) from the **at** queue.

-t *time_arg*
Specify the job time. The argument should be of the form [[*cc*]*yy*]*mmddHHMM*[.*SS*], where the parts of the argument represent the following:

ccyy	Year. If yy is specified, but cc is not, a value for yy between 69 and 99 results in a cc value of 19. Otherwise, a cc value of 20 is used.
mm	Month: a number from 1 to 12.
dd	Day: a number from 1 to 31.
HH	Hour: a number from 0 to 23.
MM	Minute: a number from 0 to 59.
SS	Second: a number from 0 to 60 (permitting a leap second), preceded by a period. The default is 0.

This is the same time format as used by touch(1).

FILES

/var/cron/atjobs directory containing job files

EXIT STATUS

The **at** utility exits with one of the following values:

0	Jobs were successfully submitted, removed, or listed.
>0	An error occurred.

EXAMPLES

Run a job at 10:25am on July 31:

```
$ at -f file -t 07311025
```

SEE ALSO

atq(1), atrm(1), sh(1), touch(1), cron(8)

NAME

bc - arbitrary-precision arithmetic language and calculator

SYNOPSIS

bc [**-cl**] [**-e** *expression*] [*file ...*]

DESCRIPTION

bc is an interactive processor for a language which resembles C but provides unlimited precision arithmetic. It takes input from any expressions on the command line and any files given, then reads the standard input.

Options available:

-c **bc** is actually a preprocessor for dc(1), which it invokes automatically, unless the **-c** (compile only) option is present. In this case the generated dc(1) instructions are sent to the standard output, instead of being interpreted by a running dc(1) process.

-e *expression*
Evaluate *expression*. If multiple **-e** options are specified, they are processed in the order given, separated by newlines.

-l Allow specification of an arbitrary precision math library. The definitions in the library are available to command line expressions.

The syntax for **bc** programs is as follows: ‘L’ means letter a-z; ‘E’ means expression; ‘S’ means statement. As a non-portable extension, it is possible to use long names in addition to single letter names. A

long name is a sequence starting with a lowercase letter followed by any number of lowercase letters and digits. The underscore character (‘_’) counts as a letter.

Comments
are enclosed in /* and */
are enclosed in # and the next newline

The newline is not part of the line comment, which in itself is a non-portable extension.

Names
simple variables: L
array elements: L [E]
The words ‘ibase’, ‘obase’, and ‘scale’
The word ‘last’ or a single dot

Other operands
arbitrarily long numbers with optional sign and decimal point
(E)
sqrt (E)
length (E) number of significant decimal digits
scale (E) number of digits right of decimal point
L (E , ... , E)

The sequence ‘\<newline><whitespace>’ is ignored within numbers.

Operators

The following arithmetic and logical operators can be used. The semantics of the operators is the same as in the C language. They are

listed in order of decreasing precedence. Operators in the same group have the same precedence.

Operator	**Associativity**	**Description**
++ --	none	increment, decrement
-	none	unary minus
^	right	power
* / %	left	multiply, divide, modulus
+ -	left	plus, minus
= += -= *= /= %= ^=	right	assignment
== <= >= != < >	none	relational
!	none	boolean not
&&	left	boolean and
||	left	boolean or

Note the following:

- The relational operators may appear in any expression. The IEEE Std 1003.1-2008 ("POSIX.1") standard only allows them in the conditional expression of an ‘if’, ‘while’ or ‘for’ statement.

- The relational operators have a lower precedence than the assignment operators. This has the consequence that the expression **a = b < c** is interpreted as **(a = b) < c**, which is probably not what the programmer intended.

- In contrast with the C language, the relational operators all have the same precedence, and are non-associative. The expression **a < b < c** will produce a syntax error.

- The boolean operators (!, && and ||) are non-portable extensions.

- The boolean not (!) operator has much lower precedence than the same operator in the C language. This has the consequence that the expression **!a < b** is interpreted as **!(a < b)**. Prudent programmers use parentheses when writing expressions involving boolean operators.

Statements

E
{ S ; ... ; S }
if (E) S
if (E) S else S
while (E) S
for (E ; E ; E) S
null statement
break
continue
quit
a string of characters, enclosed in double quotes
print E ,..., E

A string may contain any character, except double quote. The if statement with an else branch is a non-portable extension. All three E's in a for statement may be empty. This is a non-portable extension. The continue and print statements are also non-portable extensions.

The print statement takes a list of comma-separated expressions. Each expression in the list is evaluated and the computed value is printed and assigned to the variable 'last'. No trailing newline is printed. The

expression may also be a string enclosed in double quotes. Within these strings the following escape sequences may be used: ‘\a’ for bell (alert), ‘\b’ for backspace, ‘\f’ for formfeed, ‘\n’ for newline, ‘\r’ for carriage return, ‘\t’ for tab, ‘\q’ for double quote and ‘\\’ for backslash. Any other character following a backslash will be ignored. Strings will not be assigned to ‘last’.

Function definitions

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

As a non-portable extension, the opening brace of the define statement may appear on the next line. The return statement may also appear in the following forms:

```
return
return ()
return E
```

The first two are equivalent to the statement "return 0". The last form is a non-portable extension. Not specifying a return statement is equivalent to writing "return (0)".

Functions available in the math library, which is loaded by specifying the **-l** flag on the command line

```
s(x)    sine
```

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

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. The value printed is assigned to the special variable ‘last’. This is a non-portable extension. A single dot may be used as a synonym for ‘last’. Either semicolons or newlines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of dc(1). Assignments to *ibase* or *obase* set the input and output number radix respectively.

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

For example

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
```

```
                for(i=1; 1==1; i++){
                        a = a*x
                        b = b*i
                        c = a/b
                        if(c == 0) return(s)
                        s = s+c
                }
        }
```

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

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

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

```
$ bc -l -e 'scale = 500; 2 * a(2^10000)' -e quit
```

prints an approximation of pi.

COMMAND LINE EDITING

bc supports interactive command line editing, via the editline(3) library. It is enabled by default if input is from a tty. Previous lines can be recalled and edited with the arrow keys, and other GNU Emacs-style editing keys may be used as well.

The editline(3) library is configured with a *.editrc* file - refer to editrc(5) for more information.

FILES

/usr/share/misc/bc.library math library, read when the **-l** option is specified on the command line.

SEE ALSO

dc(1)

HISTORY

A **bc** command appeared in Version 6 AT&T UNIX.

NAME

cp - copy files

SYNOPSIS

cp [**-fi**] [**-R**] *source target*
cp [**-fi**] [**-R**] *source ... directory*

DESCRIPTION

In the first synopsis form, the **cp** utility copies the contents of the *source* file to the *target* file. In the second synopsis form, the contents of each named *source* file are copied to the destination *directory*. The names of the files themselves are not changed. If **cp** detects an attempt to copy a file to itself, the copy will fail.

The options are as follows:

-f For each existing destination pathname, remove it and create a new file, without prompting for confirmation, regardless of its permissions. The **-f** option overrides any previous **-i** options.

-i Write a prompt to the standard error output before copying a file that would overwrite an existing file. If the response from the standard input begins with the character ‘y’, the file copy is attempted. The **-i** option overrides any previous **-f** options.

-R If *source* designates a directory, **cp** copies the directory and the entire subtree connected at that point. Created directories have the same mode as the corresponding source directory.

In the second synopsis form, the destination specified by the *directory* operand must exist unless there is only one named *source* which is a directory and the **-R** flag is specified.

Appropriate permissions are required for file creation or overwriting.

EXIT STATUS

The **cp** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

Make a copy of file *foo* named *bar*:

```
$ cp foo bar
```

Copy a group of files to the */tmp* directory:

```
$ cp *.txt /tmp
```

Copy the directory *junk* and all of its contents (including any subdirectories) to the */tmp* directory:

```
$ cp -R junk /tmp
```

SEE ALSO

mv(1), umask(2), fts(3), symlink(7)

HISTORY

A **cp** command appeared in Version 1 AT&T UNIX.

NAME

date - display or set date and time

SYNOPSIS

date [**-ju**] [**-r** *seconds*] [**-z** *output_zone*] [**+***format*]
[[[[[[*cc*]*yy*]*mm*]*dd*]*HH*]*MM*[.*SS*]]

DESCRIPTION

The **date** utility displays the current date and time when invoked without arguments. Otherwise, depending on the options specified, **date** will set the date and time or print it in a user-defined way.

Changing the system date has some risks, as described in settimeofday(2). Only the superuser may change the date.

The options are as follows:

-j Parse the provided date and time and display the result without changing the clock.

-r *seconds*
Print out (in specified format) the date and time represented by *seconds* from the Epoch.

-u Display or set the date in UTC (Coordinated Universal) time.

-z *output_zone*
Just before printing the time, change to the specified timezone; see the description of TZ below. This can be used with **-j** to easily convert time specifications from one zone to another.

An operand with a leading plus sign (‘+’) signals a user-defined format string which specifies the format in which to display the date and time. The format string may contain any of the conversion specifications described in the strftime(3) manual page, as well as any arbitrary text. A newline (‘\n’) character is always output after the characters specified by the format string. The format string for the default display is:

%a %b %e %H:%M:%S %Z %Y

If an operand does not have a leading plus sign, it is interpreted as a value for setting the system’s notion of the current date and time. The canonical representation for setting the date and time is:

ccyy	Year. If yy is specified, but cc is not, a value for yy between 69 and 99 results in a cc value of 19. Otherwise, a cc value of 20 is used.
mm	Month: a number from 1 to 12.
dd	Day: a number from 1 to 31.
HH	Hour: a number from 0 to 23.
MM	Minute: a number from 0 to 59.
SS	Second: a number from 0 to 60 (permitting a leap second), preceded by a period.

Everything but the minute is optional.

Time changes for Daylight Saving Time, standard time, leap seconds, and leap years are handled automatically.

ENVIRONMENT

TZ The time zone to use when parsing or displaying dates. See

environ(7) for more information. If this variable is not set, the time zone is determined based on */etc/localtime*, which the administrator adjusts using the **-l** option of zic(8).

FILES

/var/log/wtmp record of date resets and time changes
/var/log/messages record of the user setting the time

EXIT STATUS

The **date** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

Display the current date and time:

```
$ date
```

Display the date using the specified format string:

```
$ date "+DATE: %Y-%m-%d%nTIME: %H:%M:%S"
DATE: 1987-11-21
TIME: 13:36:16
```

Set the date to June 13, 1985, 4:27 PM:

```
# date 198506131627
```

SEE ALSO

adjtime(2), gettimeofday(2), strftime(3), utmp(5), ntpd(8), rdate(8)

HISTORY

A **date** command appeared in Version 1 AT&T UNIX.

NAME

echo - write arguments to the standard output

SYNOPSIS

echo [**-n**] [*string* ...]

DESCRIPTION

The **echo** utility writes any specified operands, separated by single blank (‘ ’) characters and followed by a newline (‘\n’) character, to the standard output.

When no operands are given, only the newline is written. The -- operand, which generally denotes an end to option processing, is treated as part of *string*.

The options are as follows:

-n Do not print the trailing newline character.

EXIT STATUS

The **echo** utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

csh(1), ksh(1), printf(1)

NAME

find - walk a file hierarchy

SYNOPSIS

find *path ...* [*expression*]

DESCRIPTION

find recursively descends the directory tree for each *path* listed, evaluating an *expression* (composed of the "primaries" and "operators" listed below) in terms of each file in the tree. In the absence of an expression, **-print** is assumed. If an expression is given, but none of the primaries **-exec**, **-print**, or **-print0** are specified, the given expression is effectively replaced by **(** *given expression* **) -print**.

PRIMARIES

-exec *utility* [*argument ...*] ;

Execute the specified *utility*. Optional arguments may be passed to the utility. The *utility* is executed once per path. If the string "{}" appears anywhere in the utility name or the arguments it is replaced by the pathname of the current file. The expression must be terminated by a semicolon (‘;’).

-iname *pattern*

Identical to the **-name** primary except that the matching is done in a case insensitive manner.

-name *pattern*

True if the last component of the pathname being examined matches *pattern*, which may use any of the special characters documented in glob(7).

-newer *file*
True if the current file has a more recent last modification time than *file*.

-path *pattern*
True if the pathname being examined matches *pattern*, which may use any of the special characters documented in glob(7). Slashes ('/') are treated as normal characters and do not have to be matched explicitly.

-print This primary always evaluates to true. It prints the pathname of the current file to standard output, followed by a newline ('\n') character.

-print0 This primary always evaluates to true. It prints the pathname of the current file to standard output, followed by a null character, suitable for use with the **-0** option to xargs(1).

-prune This primary always evaluates to true. It causes **find** to not descend into the current file.

-type *t* True if the file is of the specified type. Possible file types are as follows:

- **b** block special
- **c** character special
- **d** directory
- **f** regular file
- **l** symbolic link
- **p** FIFO
- **s** socket

OPERATORS

The primaries may be combined using the following operators. The operators are listed in order of decreasing precedence.

(*expression* **)**
This evaluates to true if the parenthesized expression evaluates to true.

! *expression* This is the unary NOT operator. It evaluates to true if the expression is false.

expression **-a** *expression*
expression expression
The logical AND operator. As it is implied by the juxtaposition of two expressions it does not have to be specified. The expression evaluates to true if both expressions are true. The second expression is not evaluated if the first expression is false.

expression **-o** *expression*
The logical OR operator. The expression evaluates to true if either the first or the second expression is true. The second expression is not evaluated if the first expression is true.

Operators, primaries, and arguments to primaries must be separate arguments to **find**, i.e. they should be separated by whitespace.

EXIT STATUS

The **find** utility exits with a value of 0 on successful traversal of all path operands or with a value >0 if an error occurred.

EXAMPLES

Print out a list of all the files whose names end in ".c":

```
$ find / -name '*.c'
```

Find all files in */usr/src* ending in a dot and single digit, but skip directory */usr/src/gnu*:

```
$ find /usr/src -path /usr/src/gnu -prune -o -name \*.[0-9]
```

Find and remove all *.jpg and *.gif files under the current working directory:

```
$ find . \( -name \*.jpg -o -name \*.gif \) -exec rm {} \;
```

or

```
$ find . \( -name \*.jpg -o -name \*.gif \) -print0 | xargs -0r rm
```

SEE ALSO

chflags(1), chmod(1), locate(1), ls(1), whereis(1), which(1), xargs(1), stat(2), fts(3), glob(7), symlink(7)

HISTORY

A **find** command appeared in Version 1 AT&T UNIX.

CAVEATS

The special characters used by **find** are also special characters to many shell programs. In particular, the characters '*', '[', ']', '?', '(', ')', '!', '\', and ';' may have to be escaped from the shell.

NAME

grep, **egrep**, **fgrep**, - file pattern searcher

SYNOPSIS

grep [**-cilnqRsvx**] [**-e** *pattern*] [**-f** *file*] [*pattern*] [*file ...*]

DESCRIPTION

The **grep** utility searches any given input files, selecting lines that match one or more patterns. Patterns may consist of one or more lines, allowing any of the pattern lines to match a portion of the input. Each input line that matches at least one of the patterns is written to the standard output. If no file arguments are specified, the standard input is used.

The related programs are as follows:

grep Simple patterns and basic regular expressions (BREs).

egrep Extended regular expressions (EREs).

fgrep Fixed patterns only (i.e. no regular expressions).

See re_format(7) for more information on regular expressions.

The options are as follows:

-c Only a count of selected lines is written to standard output.

-e *pattern*

Specify a pattern used during the search of the input. An input line is selected if it matches any of the specified patterns.

-f *file* Read one or more newline separated patterns from *file*. Empty pattern lines match every input line. Newlines are not considered part of a pattern. If *file* is empty, nothing is matched.

-i Perform case insensitive matching. By default, **grep** is case sensitive.

-l Only the names of files containing selected lines are written to standard output. **grep** will only search a file until a match has been found, making searches potentially less expensive. Pathnames are listed once per file searched. If the standard input is searched, the string "(standard input)" is written.

-n Each output line is preceded by its relative line number in the file, starting at line 1. The line number counter is reset for each file processed. This option is ignored if **-c**, **-l**, or **-q** is specified.

-q Quiet mode. Suppress normal output.

-R Recursively search subdirectories listed.

-s Silent mode. Nonexistent and unreadable files are ignored (i.e. their error messages are suppressed).

-v Selected lines are those *not* matching any of the specified patterns.

-x Only input lines selected against an entire fixed string or regular expression are considered to be matching lines.

EXIT STATUS

The **grep** utility exits with one of the following values:

0	One or more lines were selected.
1	No lines were selected.
>1	An error occurred.

EXAMPLES

Find all occurrences of the word ‘foo’ in ‘myfile’:

```
$ grep ’foo’ myfile
```

Find all occurrences of the pattern ‘.Pp’ at the beginning of a line:

```
$ grep ’^\.Pp’ myfile
```

Find all lines in a file which do not contain the words ‘foo’ or ‘bar’:

```
$ grep -v -e ’foo’ -e ’bar’ myfile
```

Find all occurrences of 19, 20, or 25 in the file ‘calendar’:

```
$ egrep ’19|20|25’ calendar
```

SEE ALSO

ed(1), ex(1), sed(1), re_format(7)

HISTORY

A **grep** command appeared in Version 4 AT&T UNIX.

NAME

head - display first few lines of files

SYNOPSIS

head [**-n** *count*] [*file* ...]

DESCRIPTION

The **head** utility copies the first *count* lines of each specified *file* to the standard output. If no files are named, **head** copies lines from the standard input. If *count* is omitted, it defaults to 10.

The options are as follows:

-n *count*

Copy the first *count* lines of each input file to the standard output. *count* must be a positive decimal integer.

EXIT STATUS

The **head** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

Display the first 500 lines of the file *foo*:

```
$ head -n 500 foo
```

SEE ALSO

cat(1), cut(1), less(1), more(1), tail(1)

HISTORY

A **head** command appeared in 1BSD.

NAME

id - return user identity

SYNOPSIS

id [*user*]
id -G [**-n**] [*user*]
id -g | **-u** [**-nr**] [*user*]

DESCRIPTION

The **id** utility displays the user and group names and numeric IDs, of the calling process, to the standard output. If the real and effective IDs are different, both are displayed, otherwise only the real ID is displayed.

If a *user* (login name or user ID) is specified, the user and group IDs of that user are displayed. In this case, the real and effective IDs are assumed to be the same.

The options are as follows:

-G Display the different group IDs (effective, real and supplementary) as whitespace separated numbers, in no particular order.

-g Display the effective group ID as a number.

-n Display the name of the user or group ID for the **-G**, **-g** and **-u** options instead of the number. If any of the ID numbers cannot be mapped into names, the number will be displayed as usual.

-r Display the real ID for the **-g** and **-u** options instead of the effective ID.

-u Display the effective user ID as a number.

EXIT STATUS

The **id** utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

groups(1), who(1), whoami(1)

HISTORY

The historic groups(1) command is equivalent to **id -Gn** [*user*].

The historic whoami(1) command is equivalent to **id -un**.

An **id** command appeared in 4.4BSD.

NAME

jot - print sequential or random data

SYNOPSIS

jot [**-cnr**] [**-b** *word*] [**-p** *precision*] [**-s** *string*] [**-w** *word*]
[*reps* [*begin* [*end* [*s*]]]]

DESCRIPTION

jot is used to print out increasing, decreasing, random, or redundant data, usually numbers, one per line.

The options are as follows:

-b *word* Just print *word* repetitively.

-c This is an abbreviation for **-w %c**.

-n Do not print the final newline normally appended to the output.

-p *precision* Print only as many digits or characters of the data as indicated by the integer *precision*. In the absence of **-p**, the precision is the greater of the numbers *begin* and *end*. The **-p** option is overridden by whatever appears in a printf(3) conversion following **-w**.

-r Generate random data. By default, **jot** generates sequential data.

-s *string* Print data separated by *string*. Normally, newlines separate data.

-w *word* Print *word* with the generated data appended to it. Octal, hexadecimal, exponential, ASCII, zero-padded, and right-adjusted representations are possible by using the appropriate printf(3) conversion specification inside *word*, in which case the data is inserted rather than appended.

The last four arguments indicate, respectively, the maximum number of data, the lower bound, the upper bound, and the step size. While at least one of them must appear, any of the other three may be omitted, and will be considered as such if given as '-'. Any three of these arguments determines the fourth. If four are specified and the given and computed values of *reps* conflict, the lower value is used. If fewer than three are specified, defaults are assigned left to right, except for *s*, which assumes its default unless both *begin* and *end* are given.

Defaults for the four arguments are, respectively, 100, 1, 100, and 1. *reps* is expected to be an unsigned integer, and if given as zero is taken to be infinite. *begin* and *end* may be given as real numbers or as characters representing the corresponding value in ASCII. The last argument must be a real number.

Random numbers are obtained through arc4random(3). Historical versions of **jot** used *s* to seed the random number generator. This is no longer supported. The name **jot** derives in part from "iota", a function in APL.

Rounding and truncation

The **jot** utility uses double precision floating point arithmetic internally. Before printing a number, it is converted depending on the output format used.

If no output format is specified or the output format is a floating point format (‘f’, ‘e’, ‘g’, ‘E’, or ‘G’), the value is rounded using the printf(3) function, taking into account the requested precision.

If the output format is an integer format (‘c’, ‘d’, ‘o’, ‘x’, ‘u’, ‘D’, ‘O’, ‘X’, ‘U’, or ‘i’), the value is converted to an integer value by truncation.

As an illustration, consider the following command:

```
$ jot 6 1 10 0.5
1
2
2
2
3
4
```

By requesting an explicit precision of 1, the values generated before rounding can be seen. The .5 values are rounded down if the integer part is even, up otherwise.

```
$ jot -p 1 6 1 10 0.5
1.0
1.5
2.0
2.5
3.0
3.5
```

By offsetting the values slightly, the values generated by the following

command are always rounded down:

```
$ jot -p 0 6 .9999999999 10 0.5
1
1
2
2
3
3
```

Another way of achieving the same result is to force truncation by specifying an integer format:

```
$ jot -w %d 6 1 10 0.5
```

For random sequences, the output format also influences the range and distribution of the generated numbers:

```
$ jot -r 100000 1 3 | sort -n | uniq -c
24950 1
50038 2
25012 3
```

The values at the beginning and end of the interval are generated less frequently than the other values. There are several ways to solve this problem and generate evenly distributed integers:

```
$ jot -r -p 0 100000 0.5 3.5 | sort -n | uniq -c
33374 1
33363 2
33263 3
```

```
$ jot -w %d -r 100000 1 4 | sort -n | uniq -c
33306 1
33473 2
33221 3
```

Note that with random sequences, all numbers generated will be smaller than the upper bound. The largest value generated will be a tiny bit smaller than the upper bound. For floating point formats, the value is rounded as described before being printed. For integer formats, the highest value printed will be one less than the requested upper bound, because the generated value will be truncated.

EXAMPLES

Print 21 evenly spaced numbers increasing from -1 to 1:

```
$ jot 21 -1 1.00
```

Generate the ASCII character set:

```
$ jot -c 128 0
```

Generate the strings xaa through xaz:

```
$ jot -w xa%c 26 a
```

Generate 20 random 8-letter strings (note that the character ‘{’ comes after the character ‘z’ in the ASCII character set):

```
$ jot -r -c 160 a { | rs -g0 0 8
```

Infinitely many yes(1)’s may be obtained through:

```
$ jot -b yes 0
```

Thirty ed(1) substitution commands applying to lines 2, 7, 12, etc. is the result of:

```
$ jot -w %ds/old/new/ 30 2 - 5
```

Create a file containing exactly 1024 bytes:

```
$ jot -b x 512 > block
```

To set tabs four spaces apart starting from column 10 and ending in column 132, use:

```
$ expand -‘jot -s, - 10 132 4‘
```

To print all lines 80 characters or longer:

```
$ grep ‘jot -s "" -b. 80‘
```

SEE ALSO

ed(1), expand(1), rs(1), yes(1), arc4random(3), printf(3)

HISTORY

A **jot** command appeared in 4.2BSD.

NAME

kill - terminate or signal a process

SYNOPSIS

kill [**-s** *signal_name*] *pid ...*
kill -l [*exit_status*]
kill -*signal_name pid ...*
kill -*signal_number pid ...*

DESCRIPTION

The **kill** utility sends a signal to the process(es) specified by the *pid* operand(s). If no signal is specified, SIGTERM is used.

Only the superuser may send signals to other users' processes.

The options are as follows:

-l [*exit_status*]

Display the name of the signal corresponding to *exit_status*. *exit_status* may be the exit status of a command killed by a signal or a signal number.

If no operand is given, display the names of all the signals.

-s *signal_name*

A symbolic signal name specifying the signal to be sent instead of the default SIGTERM.

-*signal_name*

A symbolic signal name specifying the signal to be sent instead of the default SIGTERM.

-*signal_number*

A non-negative decimal integer specifying the signal to be sent instead of the default SIGTERM.

The following PIDs have special meanings:

-1 If superuser, broadcast the signal to all processes; otherwise, broadcast to all processes belonging to the user.

0 Send the signal to all processes whose group ID is equal to the process group ID of the sender, and for which the process has permission.

-pgid Send the signal to all processes within the specified process group.

Some of the more commonly used signals:

1 HUP (hang up)
2 INT (interrupt)
3 QUIT (quit)
6 ABRT (abort)
9 KILL (non-catchable, non-ignorable kill)
14 ALRM (alarm clock)
15 TERM (software termination signal)

For a more complete list, consult the sigaction(2) manual page.

A signal number of 0 (kill -0 pid) checks the validity of PID, to see if it exists. An exit code of 0 means that the specified process exists.

EXIT STATUS

The **kill** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

Forcibly terminate process ID 1234:

```
$ kill -9 1234
```

Send the init(8) process the hangup signal, instructing it to re-read ttys(5):

```
# kill -HUP 1
```

SEE ALSO

csh(1), ksh(1), pkill(1), ps(1), kill(2), sigaction(2)

HISTORY

A **kill** command appeared in Version 3 AT&T UNIX.

NAME

ls - list directory contents

SYNOPSIS

ls [**-1AaCcdFfgHhikLlmnopqRrSsTtux**] [*file ...*]

DESCRIPTION

For each operand that names a *file* of a type other than directory, **ls** displays its name as well as any requested, associated information. For each named directory, **ls** displays the names of files contained within that directory, as well as any requested, associated information.

If no operands are given, the contents of the current directory are displayed. If more than one operand is given, non-directory operands are displayed first; directory and non-directory operands are sorted separately and in lexicographical order. By default, **ls** lists one entry per line to standard output; the exceptions are to terminals or when the **-C**, **-m**, or **-x** options are specified.

The options are as follows:

-1 (The numeric digit "one".) Force output to be one entry per line. This is the default when output is not to a terminal.

-A List all entries except for ‘.’ and ‘..’. Always set for the superuser.

-a Include directory entries whose names begin with a dot (‘.’).

-C Force multi-column output; this is the default when output is to a terminal.

-c Use time file's status was last changed instead of last modification time for sorting (**-t**) or printing (**-l** or **-n**).

-d Directories are listed as plain files (not searched recursively) and symbolic links in the argument list are not indirected through.

-F Display a slash ('/') immediately after each pathname that is a directory, an asterisk ('*') after each that is executable, an at sign ('@') after each symbolic link, an equal sign ('=') after each socket, and a vertical bar ('|') after each that is a FIFO.

-f Output is not sorted. This option implies **-a**.

-H Follow symbolic links specified on the command line. This is the default behaviour when none of the **-d**, **-F**, or **-l** options are specified.

-h When used with a long format option, use unit suffixes: Byte, Kilobyte, Megabyte, Gigabyte, Terabyte, Petabyte, and Exabyte in order to reduce the number of digits to four or fewer using powers of 2 for sizes (K=1024, M=1048576, etc.).

-i For each file, print its inode number.

-l (The lowercase letter "ell".) List in long format (see below). A total sum of all file sizes is output on a line before the long listing. Output is one entry per line.

-m Stream output format; list files across the page, separated by

commas.

-n List in long format as in **-l**, but retain user and group IDs in a numeric format.

-p Display a slash (‘/’) immediately after each pathname that is a directory.

-q Force printing of non-graphic characters in file names as the character ‘?’; this is the default when output is to a terminal.

-R Recursively list subdirectories encountered.

-r Reverse the order of the sort to get reverse lexicographical order or the smallest or oldest entries first.

-S Sort by size, largest file first.

-t Sort by time modified (most recently modified first) before sorting the operands in lexicographical order.

-u Use file’s last access time instead of last modification time for sorting (**-t**) or printing (**-l** or **-n**).

-x Multi-column output sorted across the page rather than down the page.

It is not an error to specify more than one of the following mutually exclusive options: **-1**, **-C**, **-l**, **-m**, **-n**, and **-x**; and **-c**, **-f**, **-S**, **-t**, and **-u**. Where more than one option is specified from the same mutually exclusive group, the last option given overrides the others, except that

-l always overrides **-g**; and **-f** always overrides **-c**, **-S**, **-t**, and **-u**.

The Long Format

If the **-l** or **-n** options are given, the following information is displayed for each file: mode, number of links, owner, group, size in bytes, time of last modification ("mmm dd HH:MM"), and the pathname. In addition, for each directory whose contents are displayed, the first line displayed is the total number of blocks used by the files in the directory. Blocks are 512 bytes.

If the owner or group name is not a known user or group name, respectively, or the **-n** option is given, the numeric ID is displayed.

If the file is a character special or block special file, the major and minor device numbers for the file are displayed in the size field.

If the **-T** option is given, the time of last modification is displayed using the format "mmm dd HH:MM:SS ccyy".

If the file is a symbolic link, the pathname of the linked-to file is preceded by "->".

The file mode printed under the **-l** or **-n** options consists of the entry type, owner permissions, group permissions, and other permissions. The entry type character describes the type of file, as follows:

- **-** regular file
- **b** block special file
- **c** character special file
- **d** directory
- **l** symbolic link

p FIFO
s socket link

The next three fields are three characters each: owner permissions, group permissions, and other permissions. Each field has three character positions:

1. If **r**, the file is readable; if **-**, it is not readable.
2. If **w**, the file is writable; if **-**, it is not writable.
3. The first of the following that applies:

S If in the owner permissions, the file is not executable and set-user-ID mode is set. If in the group permissions, the file is not executable and set-group-ID mode is set.

s If in the owner permissions, the file is executable and set-user-ID mode is set. If in the group permissions, the file is executable and setgroup-ID mode is set.

x The file is executable or the directory is searchable.

- The file is neither readable, writable, executable, nor set-user-ID, nor set-group-ID, nor sticky (see below).

These next two apply only to the third character in the last group (other permissions):

T The sticky bit is set (mode 1000), but neither executable nor searchable (see chmod(1) or sticky(8)).

t The sticky bit is set (mode 1000), and is searchable or executable (see chmod(1) or sticky(8)).

EXIT STATUS

The **ls** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

List the contents of the current working directory in long format, show inode numbers, and suffix each filename with a symbol representing its file type:

```
$ ls -liF
```

List the files in */var/log*, sorting the output such that the mostly recently modified entries are printed first:

```
$ ls -lt /var/log
```

SEE ALSO

chflags(1), chmod(1), symlink(7), sticky(8)

HISTORY

An **ls** command appeared in Version 3 AT&T UNIX.

NAME

more - view files

SYNOPSIS

more [**-i**] [**-n** *number*] [**-p** *command*] [*file ...*]

DESCRIPTION

The **more** pager displays text one screenful at a time. After showing each screenful, it prompts the user for a command. Most commands scroll the text or move to a different place in the file, while some switch to another file. If no *file* is specified, or if *file* is a single dash ('-'), the standard input is used.

When showing the last line of a file, **more** displays a prompt indicating end of file and the name of the next file to examine, if any. It then waits for input from the user. Scrolling forward switches to the next file, or exits if there is none.

The options are as follows:

-i Ignore case. Upper and lower case are considered identical.

-n *number*

Page *number* of lines per screenful. By default, **more** uses the terminal window size.

-p *command*

Execute the specified **more** commands when a file is first examined (or re-examined, such as with the **:e** or **:p** commands). Multiple commands have to be concatenated into one single argument.

COMMANDS

Interactive commands for **more** are based on vi(1). Some commands may be preceded by a decimal number, called N in the descriptions below. In the following descriptions, ^X means control-X.

h Help. Display a summary of these commands.

f | ^F | SPACE
Scroll forward N lines, default one window.

b | ^B Scroll backward N lines, default one window.

d | ^D Scroll forward N lines, default one half of screen size.

u | ^U Scroll backward N lines, default one half of screen size.

j | RETURN
Scroll forward N lines, default 1.

k Scroll backward N lines, default 1.

g Go to line N in the file, default 1 (beginning of file).

G Go to line N in the file, default the end of the file.

m Followed by any lowercase letter, marks the current position with that letter.

' (Single quote.) Followed by any lowercase letter, returns to the position which was previously marked with that letter.

/pattern

Search forward in the file for the N-th line containing the pattern. N defaults to 1. The pattern is a basic regular expression. See re_format(7) for more information.

?*pattern*

Search backward in the file for the N-th line containing the pattern.

/!*pattern*

Like /, but the search is for the N-th line which does NOT contain the pattern.

?!*pattern*

Like ?, but the search is for the N-th line which does NOT contain the pattern.

n Repeat previous search, for N-th line containing the last pattern (or NOT containing the last pattern, if the previous search was /! or ?!).

N Repeat previous search in the opposite direction, for N-th line containing the last pattern (or NOT containing the last pattern, if the previous search was /! or ?!).

:e [*filename*]

Examine a new file. If *filename* is missing, the "current" file from the list of files in the command line is re-examined.

:n Examine the next file. If a number N is specified, the N-th next file is examined.

:p Examine the previous file. If a number N is specified, the N-th previous file is examined.

v Invokes an editor to edit the current file being viewed. The editor is taken from the environment variable EDITOR, or defaults to vi(1).

q | :q | ZZ
Exits **more**.

ENVIRONMENT

EDITOR
Specifies the default editor. If not set, vi(1) is used.

TERM Specifies the terminal type. Used by **more** to get the terminal characteristics necessary to manipulate the screen.

EXIT STATUS

The **more** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

Examine the ends of all files in the current directory, showing line and byte counts for each:

```
$ more -p G= *
```

SEE ALSO

less(1), vi(1), re_format(7)

HISTORY

A **more** command appeared in 3.0BSD.

NAME

nc - arbitrary TCP and UDP connections and listens

SYNOPSIS

nc [**-46DdFhklNnrStUuvz**] [**-I** *length*] [**-i** *interval*] [**-O** *length*]
[**-P** *proxy_username*] [**-p** *source_port*] [**-s** *source*] [**-T** *toskeyword*]
[**-V** *rtable*] [**-w** *timeout*] [**-X** *proxy_protocol*]
[**-x** *proxy_address*[:*port*]] [*destination*] [*port*]

DESCRIPTION

The **nc** (or **netcat**) utility is used for just about anything under the sun involving TCP, UDP, or UNIX-domain sockets. It can open TCP connections, send UDP packets, listen on arbitrary TCP and UDP ports, do port scanning, and deal with both IPv4 and IPv6. Unlike telnet(1), **nc** scripts nicely, and separates error messages onto standard error instead of sending them to standard output, as telnet(1) does with some.

Common uses include:

- simple TCP proxies
- shell-script based HTTP clients and servers
- network daemon testing
- a SOCKS or HTTP ProxyCommand for ssh(1)
- and much, much more

The options are as follows:

-4 Forces **nc** to use IPv4 addresses only.

-6 Forces **nc** to use IPv6 addresses only.

-D Enable debugging on the socket.

-d Do not attempt to read from stdin.

-F Pass the first connected socket using sendmsg(2) to stdout and exit. This is useful in conjunction with **-X** to have **nc** perform connection setup with a proxy but then leave the rest of the connection to another program (e.g. ssh(1) using the ssh_config(5) **ProxyUseFdPass** option).

-h Prints out **nc** help.

-I *length*
Specifies the size of the TCP receive buffer.

-i *interval*
Specifies a delay time interval between lines of text sent and received. Also causes a delay time between connections to multiple ports.

-k Forces **nc** to stay listening for another connection after its current connection is completed. It is an error to use this option without the **-l** option. When used together with the **-u** option, the server socket is not connected and it can receive UDP datagrams from multiple hosts.

-l Used to specify that **nc** should listen for an incoming connection rather than initiate a connection to a remote host. It is an error to use this option in conjunction with the **-p**, **-s**, or **-z** options. Additionally, any timeouts specified with the **-w** option are ignored.

-N shutdown(2) the network socket after EOF on the input. Some servers require this to finish their work.

-n Do not do any DNS or service lookups on any specified addresses, hostnames or ports.

-O *length*
Specifies the size of the TCP send buffer.

-P *proxy_username*
Specifies a username to present to a proxy server that requires authentication. If no username is specified then authentication will not be attempted. Proxy authentication is only supported for HTTP CONNECT proxies at present.

-p *source_port*
Specifies the source port **nc** should use, subject to privilege restrictions and availability. It is an error to use this option in conjunction with the **-l** option.

-r Specifies that source and/or destination ports should be chosen randomly instead of sequentially within a range or in the order that the system assigns them.

-S Enables the RFC 2385 TCP MD5 signature option.

-s *source*
Specifies the IP of the interface which is used to send the packets. For UNIX-domain datagram sockets, specifies the local temporary socket file to create and use so that datagrams can be received. It is an error to use this option in

conjunction with the **-l** option.

-T *toskeyword*
Change IPv4 TOS value. *toskeyword* may be one of *critical*, *inetcontrol*, *lowdelay*, *netcontrol*, *throughput*, *reliability*, or one of the DiffServ Code Points: *ef*, *af11 ... af43*, *cs0 ... cs7*; or a number in either hex or decimal.

-t Causes **nc** to send RFC 854 DON'T and WON'T responses to RFC 854 DO and WILL requests. This makes it possible to use **nc** to script telnet sessions.

-U Specifies to use UNIX-domain sockets.

-u Use UDP instead of the default option of TCP. For UNIX-domain sockets, use a datagram socket instead of a stream socket. If a UNIX-domain socket is used, a temporary receiving socket is created in */tmp* unless the **-s** flag is given.

-V *rtable*
Set the routing table to be used.

-v Have **nc** give more verbose output.

-w *timeout*
Connections which cannot be established or are idle timeout after *timeout* seconds. The **-w** flag has no effect on the **-l** option, i.e. **nc** will listen forever for a connection, with or without the **-w** flag. The default is no timeout.

-X *proxy_protocol*

Requests that **nc** should use the specified protocol when talking to the proxy server. Supported protocols are "4" (SOCKS v.4), "5" (SOCKS v.5) and "connect" (HTTPS proxy). If the protocol is not specified, SOCKS version 5 is used.

-x *proxy_address*[:*port*]
Requests that **nc** should connect to *destination* using a proxy at *proxy_address* and *port*. If *port* is not specified, the well-known port for the proxy protocol is used (1080 for SOCKS, 3128 for HTTPS).

-z Specifies that **nc** should just scan for listening daemons, without sending any data to them. It is an error to use this option in conjunction with the **-l** option.

destination can be a numerical IP address or a symbolic hostname (unless the **-n** option is given). In general, a destination must be specified, unless the **-l** option is given (in which case the local host is used). For UNIX-domain sockets, a destination is required and is the socket path to connect to (or listen on if the **-l** option is given).

port can be a single integer or a range of ports. Ranges are in the form nn-mm. In general, a destination port must be specified, unless the **-U** option is given.

CLIENT/SERVER MODEL

It is quite simple to build a very basic client/server model using **nc**. On one console, start **nc** listening on a specific port for a connection. For example:

```
$ nc -l 1234
```

nc is now listening on port 1234 for a connection. On a second console (or a second machine), connect to the machine and port being listened on:

```
$ nc 127.0.0.1 1234
```

There should now be a connection between the ports. Anything typed at the second console will be concatenated to the first, and vice-versa. After the connection has been set up, **nc** does not really care which side is being used as a 'server' and which side is being used as a 'client'. The connection may be terminated using an EOF ('^D').

DATA TRANSFER

The example in the previous section can be expanded to build a basic data transfer model. Any information input into one end of the connection will be output to the other end, and input and output can be easily captured in order to emulate file transfer.

Start by using **nc** to listen on a specific port, with output captured into a file:

```
$ nc -l 1234 > filename.out
```

Using a second machine, connect to the listening **nc** process, feeding it the file which is to be transferred:

```
$ nc -N host.example.com 1234 < filename.in
```

After the file has been transferred, the connection will close

automatically.

TALKING TO SERVERS

It is sometimes useful to talk to servers "by hand" rather than through a user interface. It can aid in troubleshooting, when it might be necessary to verify what data a server is sending in response to commands issued by the client. For example, to retrieve the home page of a web site:

```
$ printf "GET / HTTP/1.0\r\n\r\n" | nc host.example.com 80
```

Note that this also displays the headers sent by the web server. They can be filtered, using a tool such as sed(1), if necessary.

More complicated examples can be built up when the user knows the format of requests required by the server. As another example, an email may be submitted to an SMTP server using:

```
$ nc localhost 25 << EOF
HELO host.example.com
MAIL FROM:<user@host.example.com>
RCPT TO:<user2@host.example.com>
DATA
Body of email.
.
QUIT
EOF
```

PORT SCANNING

It may be useful to know which ports are open and running services on a target machine. The **-z** flag can be used to tell **nc** to report open

ports, rather than initiate a connection. For example:

```
$ nc -z host.example.com 20-30
Connection to host.example.com 22 port [tcp/ssh] succeeded!
Connection to host.example.com 25 port [tcp/smtp] succeeded!
```

The port range was specified to limit the search to ports 20 - 30.

Alternatively, it might be useful to know which server software is running, and which versions. This information is often contained within the greeting banners. In order to retrieve these, it is necessary to first make a connection, and then break the connection when the banner has been retrieved. This can be accomplished by specifying a small timeout with the **-w** flag, or perhaps by issuing a "QUIT" command to the server:

```
$ echo "QUIT" | nc host.example.com 20-30
SSH-1.99-OpenSSH_3.6.1p2
Protocol mismatch.
220 host.example.com IMS SMTP Receiver Version 0.84 Ready
```

EXAMPLES

Open a TCP connection to port 42 of host.example.com, using port 31337 as the source port, with a timeout of 5 seconds:

```
$ nc -p 31337 -w 5 host.example.com 42
```

Open a UDP connection to port 53 of host.example.com:

```
$ nc -u host.example.com 53
```

Open a TCP connection to port 42 of host.example.com using 10.1.2.3 as the IP for the local end of the connection:

```
$ nc -s 10.1.2.3 host.example.com 42
```

Create and listen on a UNIX-domain stream socket:

```
$ nc -lU /var/tmp/dsocket
```

Connect to port 42 of host.example.com via an HTTP proxy at 10.2.3.4, port 8080. This example could also be used by ssh(1); see the **ProxyCommand** directive in ssh_config(5) for more information.

```
$ nc -x10.2.3.4:8080 -Xconnect host.example.com 42
```

The same example again, this time enabling proxy authentication with username "ruser" if the proxy requires it:

```
$ nc -x10.2.3.4:8080 -Xconnect -Pruser host.example.com 42
```

SEE ALSO

cat(1), ssh(1)

CAVEATS

UDP port scans using the **-uz** combination of flags will always report success irrespective of the target machine's state. However, in conjunction with a traffic sniffer either on the target machine or an intermediary device, the **-uz** combination could be useful for communications diagnostics. Note that the amount of UDP traffic generated may be limited either due to hardware resources and/or configuration settings.

NAME

od - octal, decimal, hexadecimal, ascii dump

SYNOPSIS

od [**-bcdhosv**] [**-A** *base*] [**-j** *offset*] [**-N** *length*] [*file ...*]

DESCRIPTION

The **od** utility sequentially copies the input from each specified file, or the standard input if no files are specified, to the standard output, transforming the data according to the options given.

If no options are specified, the default display is equivalent to specifying the **-o** option.

The options are as follows:

-A *base*

Specify the input address base. The argument *base* may be one of **o**, **d**, **x**, or **n**, which specify octal, decimal, hexadecimal addresses or no address, respectively.

-b *One-byte octal display*. Display the input offset in octal, followed by sixteen space-separated, three column, zero-filled, bytes of input data, in octal, per line. This is the default output style if no other is selected.

-c *One-byte character display*. Display the input offset in octal, followed by sixteen space-separated, three column, space-filled, characters of input data, per line. Control characters are printed as ‘C’ style escapes, or as three octal digits, if no ‘C’ escape exists for the character.

-d *Two-byte unsigned decimal display*. Display the input offset in octal, followed by eight space-separated, five column, zero-filled, two-byte units of input data, in unsigned decimal, per line.

-h *Two-byte hexadecimal display*. Display the input offset in octal, followed by eight space-separated, four column, zero filled, two-byte units of input data, in hexadecimal, per line.

-j *offset*

Skip *offset* bytes from the beginning of the input. By default, *offset* is interpreted as a decimal number. With a leading **0x** or **0X**, *offset* is interpreted as a hexadecimal number, otherwise, with a leading **0**, *offset* is interpreted as an octal number. Appending the character **b**, **k**, or **m** to *offset* causes it to be interpreted as a multiple of 512, 1024, or 1048576, respectively.

-N *length*

Interpret only *length* bytes of input. *length* is interpreted as a decimal number.

-o *Two-byte octal display*. Display the input offset in octal, followed by eight space-separated, six column, zero-filled, two-byte units of input data, in octal, per line.

-s *Two-byte signed decimal display*. Display the input offset in octal, followed by eight space-separated, five column, zero-filled, two-byte units of input data, in signed decimal, per line.

-v The **-v** option causes **od** to display all input data. Without the **-v** option, any number of groups of output lines, which would be identical to the immediately preceding group of output lines (except for the input offsets), are replaced with a line comprised of a single asterisk (‘*’).

EXIT STATUS

The **od** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

Display two-byte hexadecimal output, but skip the first 16 bytes:

```
$ od -hv -j 16 file
```

SEE ALSO

hexdump(1), ascii(7)

HISTORY

An **od** command appeared in Version 1 AT&T UNIX.

NAME

ps - display process status

SYNOPSIS

ps [**-AaCceHhjkLlmrSTuvwx**] [**-M** *core*] [**-N** *system*] [**-O** *fmt*] [**-o** *fmt*] [**-p** *pid*] [**-t** *tty*] [**-U** *username*] [**-W** *swap*]

DESCRIPTION

The **ps** utility displays information about active processes. When given no options, **ps** prints information about processes of the current user that have a controlling terminal.

The information displayed is selected based on a set of keywords (and for even more control, see the **-L**, **-O**, and **-o** options). The default output format includes, for each process, the process's ID, controlling terminal, state, CPU time (including both user and system time), and associated command.

The options are as follows:

-A Display information about processes for all users, including those without controlling terminals.

-a Display information about processes for all users with controlling terminals.

-C Change the way the CPU percentage is calculated by using a "raw" CPU calculation that ignores "resident" time (this normally has no effect).

-c Do not display full command with arguments, but only the

executable name. This may be somewhat confusing; for example, all sh(1) scripts will show as "sh".

-e Display the environment as well.

-H Also display information about kernel visible threads.

-h Repeat the information header as often as necessary to guarantee one header per page of information.

-j Print information associated with the following keywords: user, pid, ppid, pgid, sess, jobc, state, tt, time, and command.

-k Also display information about kernel threads.

-L List the set of available keywords. This option should not be specified with other options.

-l Display information associated with the following keywords: uid, pid, ppid, cpu, pri, nice, vsz, rss, wchan, state, tt, time, and command.

-M *core*

Extract values associated with the name list from the specified core instead of the running kernel.

-m Sort by memory usage, instead of by start time ID.

-N *system*

Extract the name list from the specified system instead of the running kernel.

-O *fmt* Add the information associated with the space or comma separated list of keywords specified, after the process ID, in the default information display. Keywords may be appended with an equals sign ('=') and a string. This causes the printed header to use the specified string instead of the standard header.

-o *fmt* Display information associated with the space or comma separated list of keywords specified. Keywords may be appended with an equals sign ('=') and a string. This causes the printed header to use the specified string instead of the standard header.

-p *pid* Display information associated with the specified process ID.

-r Sort by current CPU usage, instead of by start time ID.

-S Change the way the process time is calculated by summing all exited children to their parent process.

-T Display information about processes attached to the device associated with the standard input.

-t *tty* Display information about processes attached to the specified terminal device.

-U *username*
Display the processes belonging to the specified *username*.

-u Display information associated with the following keywords: user, pid, %cpu, %mem, vsz, rss, tt, state, start, time, and

command. The **-u** option implies the **-r** option.

-v Display information associated with the following keywords: pid, state, time, sl, re, pagein, vsz, rss, lim, tsiz, %cpu, %mem, and command. The **-v** option implies the **-m** option.

-W *swap*
When not using the running kernel, extract swap information from the specified file.

-w Use 132 columns to display information, instead of the default, which is the window size. If the **-w** option is specified more than once, **ps** will use as many columns as necessary without regard for window size.

-x Display information about processes without controlling terminals.

KEYWORDS

The following is a complete list of the available keywords and their meanings. Several of them have aliases, which are also noted.

%cpu Alias: **pcpu**. The CPU utilization of the process; this is a decaying average over up to a minute of previous (real) time. Since the time base over which this is computed varies (since processes may be very young) it is possible for the sum of all **%cpu** fields to exceed 100%.

%mem Alias: **pmem**. The percentage of real memory used by this process.

acflag Alias: **acflg**. Accounting flag.

command Alias: **args**. Command and arguments.

cpu Short-term CPU usage factor (for scheduling).

cpuid CPU ID (zero on single processor systems).

cwd Current working directory.

dsiz Data size, in Kilobytes.

emul Name of system call emulation environment.

flags Alias: **f**. The thread flags (in hexadecimal), as defined in the include file <*sys/proc.h*>:

P_INKTR 0x1 writing ktrace(2) record
P_PROFPEND 0x2 this thread needs SIGPROF
P_ALRMPEND 0x4 this thread needs SIGVTALRM
P_SIGSUSPEND 0x8 need to restore before-suspend mask
P_CANTSLEEP 0x10 this thread is not permitted to sleep
P_SELECT 0x40 selecting; wakeup/waiting danger
P_SINTR 0x80 sleep is interruptible
P_SYSTEM 0x200 system process: no sigs, stats, or swapping
P_TIMEOUT 0x400 timing out during sleep
P_WEXIT 0x2000 working on exiting
P_OWEUPC 0x8000 profiling sample needs recording
P_SUSPSINGLE 0x80000 need to suspend for single threading
P_SYSTRACE 0x400000 systrace(4) policy is active

P_CONTINUED 0x800000 thread has continued after a stop
P_THREAD 0x4000000 not the original thread
P_SUSPSIG 0x8000000 stopped because of a signal
P_SOFTDEP 0x10000000 stuck processing softdep worklist
P_CPUPEG 0x40000000 do not move to another cpu

gid Effective group.

group Text name of effective group ID.

inblk Alias: **inblock**. Total blocks read.

jobc Job control count.

ktrace Tracing flags.

ktracep Tracing vnode.

lim The soft limit on memory used, specified via a call to setrlimit(2).

logname Alias: **login**. Login name of user who started the process.

lstart The exact time the command started, using the "%c" format described in strftime(3).

majflt Total page faults.

maxrss Maximum resident set size (in 1024 byte units).

minflt Total page reclaims.

msgrcv Total messages received (reads from pipes/sockets).

msgsnd Total messages sent (writes on pipes/sockets).

nice Alias: **ni**. The process scheduling increment (see setpriority(2)).

nivcsw Total involuntary context switches.

nsigs Alias: **nsignals**. Total signals taken.

nswap Total swaps in/out.

nvcsw Total voluntary context switches.

nwchan Wait channel (as an address).

oublk Alias: **oublock**. Total blocks written.

p_ru Resource usage (valid only for zombie processes).

paddr Swap address.

pagein Pageins (same as **majflt**).

pgid Process group number.

pid Process ID.

ppid Parent process ID.

pri Scheduling priority.

procflags The process flags (in hexadecimal), as defined in the include file <*sys/proc.h*>:

PS_CONTROLT 0x1 process has a controlling terminal
PS_EXEC 0x2 process called exec(3)
PS_INEXEC 0x4 process is doing an exec right now
PS_EXITING 0x8 process is exiting
PS_SUGID 0x10 process had set ID privileges since last exec
PS_SUGIDEXEC 0x20 last exec(3) was set[ug]id
PS_PPWAIT 0x40 parent is waiting for process to exec/exit
PS_ISPWAIT 0x80 process is parent of PPWAIT child
PS_PROFIL 0x100 process has started profiling
PS_TRACED 0x200 process is being traced
PS_WAITED 0x400 debugging process has waited for child
PS_COREDUMP 0x800 busy coredumping
PS_SINGLEEXIT 0x1000 other threads must die
PS_SINGLEUNWIND 0x2000 other threads must unwind
PS_NOZOMBIE 0x4000 pid 1 waits for me instead of dad
PS_STOPPED 0x8000 just stopped, need to send SIGCHLD

PS_SYSTEM 0x10000 No signals, stats or swapping
PS_EMBRYO 0x20000 New process, not yet fledged
PS_ZOMBIE 0x40000 Dead and ready to be waited for
PS_NOBROADCASTKILL 0x80000 Process excluded from ki

re Core residency time (in seconds; 127 = infinity).

rgid Real group ID.

rgroup Text name of real group ID.

rlink Reverse link on run queue, or 0.

rss The real memory (resident set) size of the process (in 1024 byte units).

rsz Alias: **rssize**. Resident set size + (text size / text use count).

rtable Routing table.

ruid Real user ID.

ruser User name (from **ruid**).

sess Session pointer.

sig Alias: **pending**. Pending signals.

sigcatch Alias: **caught**. Caught signals.

sigignore Alias: **ignored**. Ignored signals.

sigmask Alias: **blocked**. Blocked signals.

sl Sleep time (in seconds; 127 = infinity).

ssiz Stack size, in Kilobytes.

start Alias: **etime**. The time the command started. If the command started less than 24 hours ago, the start time is displayed using the "%l:%M%p" format described in strftime(3). If the command started less than 7 days ago, the start time is displayed using the "%a%I%p" format. Otherwise, the start time is displayed using the "%e%b%y" format.

state Alias: **stat**. The state is given by a sequence of letters, for example, "RWN". The first letter indicates the run state of the process:

D	Marks a process in disk (or other short term, uninterruptible) wait.
I	Marks a process that is idle (sleeping for longer than about 20 seconds).
R	Marks a runnable process.
S	Marks a process that is sleeping for less than about 20 seconds.
T	Marks a stopped process.
Z	Marks a dead process (a "zombie").

Additional characters after these, if any, indicate

additional state information:

+	The process is in the foreground process group of its control terminal.
<	The process has a raised CPU scheduling priority (see setpriority(2)).
>	The process has specified a soft limit on memory requirements and is currently exceeding that limit; such a process is (necessarily) not swapped.
E	The process is trying to exit.
K	The process is a kernel thread.
N	The process has a reduced CPU scheduling priority.
s	The process is a session leader.
V	The process is suspended during a vfork(2).
X	The process is being traced or debugged.
x	The process is being monitored by systrace(1).
/*n*	On multiprocessor machines, specifies processor number *n*.

svgid Saved GID from a setgid executable.

svuid Saved UID from a setuid executable.

tdev Control terminal device number.

time Alias: **cputime**. Accumulated CPU time, user + system.

tpgid Control terminal process group ID.

tsess Control terminal session pointer.

tsiz Text size, in Kilobytes.

tt An abbreviation for the pathname of the controlling terminal, if any. The abbreviation consists of the two letters following "/dev/tty", or, for the console, "co". This is followed by a '-' if the process can no longer reach that controlling terminal (i.e. it has been revoked).

tty Full name of control terminal.

ucomm Alias: **comm**. Name to be used for accounting.

uid Effective user ID.

upr Alias: **usrpri**. Scheduling priority on return from system call.

user User name (from **uid**).

vsz Alias: **vsize**. Virtual size, in Kilobytes.

wchan The event (an address in the system) on which a process waits. When printed numerically, the initial part of the address is trimmed off and the result is printed in hex; for example, 0x80324000 prints as 324000.

xstat Exit or stop status (valid only for stopped or zombie process).

ENVIRONMENT

The following environment variables affect the execution of **ps**:

COLUMNS If set, specifies the user's preferred output width in column positions. By default, **ps** attempts to automatically determine the terminal width.

TZ The time zone to use when displaying dates. See environ(7) for more information.

FILES

/dev	special files and device names
/var/db/kvm_bsd.db	system namelist database
/var/run/dev.db	*/dev* name database

EXIT STATUS

The **ps** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

Display information on all system processes:

```
$ ps -auxw
```

SEE ALSO

fstat(1), kill(1), netstat(1), pgrep(1), pkill(1), procmap(1), systat(1), top(1), w(1), kvm(3), strftime(3), dev_mkdb(8), iostat(8), pstat(8), vmstat(8)

HISTORY

A **ps** command appeared in Version 3 AT&T UNIX in section 8 of the manual.

CAVEATS

When printing using the **command** keyword, a process that has exited and has a parent that has not yet waited for the process (in other words, a zombie) is listed as "<defunct>", and a process which is blocked while trying to exit is listed as "<exiting>". **ps** makes an educated guess as to the file name and arguments given when the process was created by examining memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be depended on too much. The **ucomm** (accounting) keyword can, however, be depended on.

BUGS

Since **ps** cannot run faster than the system and is run as any other scheduled process, the information it displays can never be exact.

NAME

quota - display disk usage and limits

SYNOPSIS

quota [**-q** | **-v**] [**-gu**]
quota [**-q** | **-v**] **-g** *group* ...
quota [**-q** | **-v**] **-u** *user* ...

DESCRIPTION

quota displays users' disk usage and limits. By default only the user quotas are printed.

The options are as follows:

-g Print group quotas for the group of which the user is a member.

-q Print a more terse message, containing only information on filesystems where usage is over quota.

-u Print user quotas for the user. This flag is equivalent to the default.

-v **quota** will display quotas on filesystems where no storage is allocated.

Specifying both **-g** and **-u** displays both the user quotas and the group quotas (for the user).

Only the superuser may use the **-u** flag and the optional *user* argument to view the limits of other users. Non-superusers can use the **-g** flag

and optional *group* argument to view only the limits of groups of which they are members.

The **-q** flag takes precedence over the **-v** flag.

quota tries to report the quotas of all mounted filesystems. If the filesystem is mounted via NFS, it will attempt to contact the rpc.rquotad(8) daemon on the NFS server. For FFS filesystems, quotas must be turned on in */etc/fstab*.

FILES

quota.user	located at the filesystem root with user quotas
quota.group	located at the filesystem root with group quotas
/etc/fstab	to find filesystem names and locations

EXIT STATUS

The **quota** utility exits 0 on success, and with a non-zero value if one or more filesystems are over quota.

SEE ALSO

quotactl(2), fstab(5), edquota(8), quotacheck(8), quotaon(8), repquota(8), rpc.rquotad(8)

HISTORY

A **quota** command appeared in 4.2BSD.

NAME

rm - remove directory entries

SYNOPSIS

rm [**-dfiPRr**] *file* ...

DESCRIPTION

The **rm** utility attempts to remove the non-directory type files specified on the command line. If the permissions of the file do not permit writing, and the standard input device is a terminal, the user is prompted (on the standard error output) for confirmation.

The options are as follows:

-d Attempt to remove directories as well as other types of files.

-f Attempt to remove the files without prompting for confirmation, regardless of the file's permissions.

-i Request confirmation before attempting to remove each file, regardless of the file's permissions, or whether or not the standard input device is a terminal.

-P Overwrite regular files before deleting them. Files are overwritten once with a random pattern. Files with multiple links will be unlinked but not overwritten.

-R Attempt to remove the file hierarchy rooted in each file argument. The **-R** option implies the **-d** option.

-r Equivalent to **-R**.

The **rm** utility removes symbolic links, not the files referenced by the links.

It is an error to attempt to remove the files "." or "..". It is forbidden to remove the file ".." merely to avoid the antisocial consequences of inadvertently doing something like "**rm -r .***".

EXIT STATUS

The **rm** utility exits 0 if all of the named files or file hierarchies were removed, or if the **-f** option was specified and all of the existing files or file hierarchies were removed. If an error occurs, **rm** exits with a value >0.

EXAMPLES

Recursively remove all files contained within the *foobar* directory hierarchy:

```
$ rm -R foobar
```

Remove the file *-Foo* from the current working directory:

```
$ rm -- -Foo
```

SEE ALSO

rmdir(1), unlink(2), fts(3), symlink(7)

HISTORY

An **rm** command appeared in Version 1 AT&T UNIX.

The interactive mode used to be a **dsw** command, a carryover from the ancient past with an amusing etymology.

NAME

sort - sort, merge, or sequence check text and binary files

SYNOPSIS

sort [**-bcCdfimnr**] [**-o** *output*] [**-t** *char*] [*file ...*]

DESCRIPTION

The **sort** utility sorts text and binary files by lines. A line is a record separated from the subsequent record by a newline character. A record can contain any printable or non-printable characters. Comparisons are based on one or more sort keys extracted from each line of input and are performed lexicographically. By default, if keys are not given, **sort** uses entire lines for comparison.

If no *file* is specified, or if *file* is '-', the standard input is used.

The options are as follows:

-C Check that the single input file is sorted. If it is, exit 0; if it's not, exit 1. In either case, produce no output.

-c Like **-C**, but additionally write a message to *stderr* if the input file is not sorted.

-m Merge only; the input files are assumed to be pre-sorted. If they are not sorted, the output order is undefined.

-o *output*

Write the output to the file *output* instead of the standard output. This file can be the same as one of the input files.

The following options override the default ordering rules:

-b Ignore leading blank characters when comparing lines.

-d Consider only blank spaces and alphanumeric characters in comparisons.

-f Consider all lowercase characters that have uppercase equivalents to be the same for purposes of comparison.

-i Ignore all non-printable characters.

-n An initial numeric string, consisting of optional blank space, optional minus sign, and zero or more digits (including decimal point) is sorted by arithmetic value. Leading blank characters are ignored.

-r Sort in reverse order.

The following option alters the treatment of field separators:

-t *char*

Use *char* as the field separator character. If **-t** is not specified, the default field separator is a sequence of blank-space characters, and consecutive blank spaces do *not* delimit an empty field; further, the initial blank space *is* considered part of a field when determining key offsets. To use NUL as field separator, use **-t** '\0'.

EXIT STATUS

The **sort** utility exits with one of the following values:

0	Successfully sorted the input files or if used with **-C** or **-c**, the input file already met the sorting criteria.
1	On disorder (or non-uniqueness) with the **-C** or **-c** options.
2	An error occurred.

SEE ALSO

comm(1), join(1), uniq(1)

HISTORY

A **sort** command appeared in Version 3 AT&T UNIX.

NAME

tail - display the last part of a file

SYNOPSIS

tail [**-f** | **-r**] [**-b** *number* | **-c** *number* | **-n** *number* | **-***number*] [*file ...*]

DESCRIPTION

The **tail** utility displays the contents of *file* or, by default, its standard input, to the standard output.

The display begins at a byte, line, or 512-byte block location in the input. Numbers having a leading plus ('+') sign are relative to the beginning of the input, for example, **-c +2** starts the display at the second byte of the input. Numbers having a leading minus ('-') sign or no explicit sign are relative to the end of the input, for example, **-n 2** displays the last two lines of the input. The default starting location is **-n 10**, or the last 10 lines of the input.

The options are as follows:

-b *number*

The location is *number* 512-byte blocks.

-c *number*

The location is *number* bytes.

-f Do not stop when end-of-file is reached; instead, wait for additional data to be appended to the input. If the file is replaced (i.e., the inode number changes), **tail** will reopen the file and continue. If the file is truncated, **tail** will reset its position to the beginning. This makes **tail** more useful for

watching log files that may get rotated. The **-f** option is ignored if there are no *file* arguments and the standard input is a pipe or a FIFO.

-n *number* | **-***number*
The location is *number* lines.

-r The **-r** option causes the input to be displayed in reverse order, by line. Additionally, this option changes the meaning of the **-b**, **-c**, and **-n** options. When the **-r** option is specified, these options specify the number of bytes, lines or 512-byte blocks to display, instead of the bytes, lines, or blocks from the beginning or end of the input from which to begin the display. The default for the **-r** option is to display all of the input.

If more than a single file is specified, each file is preceded by a header consisting of the string "==> XXX <==" where "XXX" is the name of the file.

EXIT STATUS

The **tail** utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

To display the last 500 lines of the file *foo*:

```
$ tail -500 foo
```

Keep */var/log/messages* open, displaying to the standard output anything appended to the file:

```
$ tail -f /var/log/messages
```

SEE ALSO

cat(1), head(1), sed(1)

HISTORY

A **tail** command appeared in Version 7 AT&T UNIX.

NAME

uniq - report or filter out repeated lines in a file

SYNOPSIS

uniq [**-c**] [**-d** | **-u**] [**-f** *fields*] [**-s** *chars*] [*input_file* [*output_file*]]

DESCRIPTION

The **uniq** utility reads the standard input comparing adjacent lines and writes a copy of each unique input line to the standard output. The second and succeeding copies of identical adjacent input lines are not written. Repeated lines in the input will not be detected if they are not adjacent, so it may be necessary to sort the files first.

The options are as follows:

-c Precede each output line with the count of the number of times the line occurred in the input, followed by a single space.

-d Only output lines which have duplicates.

-f *fields*

Ignore the first *fields* in each input line when doing comparisons. A field is a string of non-blank characters separated from adjacent fields by blanks, with blanks considered part of the following field. Field numbers are one based, i.e., the first field is field one.

-s *chars*

Ignore the first *chars* characters in each input line when doing comparisons. If specified in conjunction with the **-f** option,

the first *chars* characters after the first *fields* fields will be ignored. Character numbers are one based, i.e., the first character is character one.

-u Only output lines which are unique.

If additional arguments are specified on the command line, the first such argument is used as the name of an input file, the second is used as the name of an output file. A file name of '-' denotes the standard input or the standard output (depending on its position on the command line).

EXIT STATUS

The **uniq** utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

sort(1)

NAME

ex, **vi**, **view** - text editors

SYNOPSIS

ex [**-FRrSsv**] [**-c** *cmd*] [**-t** *tag*] [**-w** *size*] [*file ...*]
vi [**-eFRrS**] [**-c** *cmd*] [**-t** *tag*] [**-w** *size*] [*file ...*]
view [**-eFrS**] [**-c** *cmd*] [**-t** *tag*] [**-w** *size*] [*file ...*]

DESCRIPTION

ex is a line-oriented text editor; **vi** is a screen-oriented text editor. **ex** and **vi** are different interfaces to the same program, and it is possible to switch back and forth during an edit session. **view** is the equivalent of using the **-R** (read-only) option of **vi**.

This manual page is the one provided with the **nex/nvi** versions of the **ex/vi** text editors. **nex/nvi** are intended as bug-for-bug compatible replacements for the original Fourth Berkeley Software Distribution (4BSD) **ex** and **vi** programs. For the rest of this manual page, **nex/nvi** is used only when it's necessary to distinguish it from the historic implementations of **ex/vi**.

This manual page is intended for users already familiar with **ex/vi**. Anyone else should almost certainly read a good tutorial on the editor before this manual page. If you're in an unfamiliar environment, and you absolutely have to get work done immediately, read the section after the options description, entitled *FAST STARTUP*. It's probably enough to get you going.

The following options are available:

-c *cmd* Execute *cmd* on the first file loaded. Particularly useful for

initial positioning in the file, although *cmd* is not limited to positioning commands. This is the POSIX 1003.2 interface for the historic "+cmd" syntax. **nex/nvi** supports both the old and new syntax.

-e Start editing in ex mode, as if the command name were **ex**.

-F Don't copy the entire file when first starting to edit. (The default is to make a copy in case someone else modifies the file during your edit session.)

-R Start editing in read-only mode, as if the command name was **view**, or the **readonly** option was set.

-r Recover the specified files or, if no files are specified, list the files that could be recovered. If no recoverable files by the specified name exist, the file is edited as if the **-r** option had not been specified.

-S Run with the **secure** edit option set, disallowing all access to external programs.

-s Enter batch mode; applicable only to **ex** edit sessions. Batch mode is useful when running **ex** scripts. Prompts, informative messages and other user oriented messages are turned off, and no startup files or environment variables are read. This is the POSIX 1003.2 interface for the historic "-" argument. **nex/nvi** supports both the old and new syntax.

-t *tag* Start editing at the specified *tag* (see ctags(1)).

-v Start editing in vi mode, as if the command name was **vi**.

-w *size* Set the initial window size to the specified number of lines.

Command input for **ex/vi** is read from the standard input. In the **vi** interface, it is an error if standard input is not a terminal. In the **ex** interface, if standard input is not a terminal, **ex** will read commands from it regardless; however, the session will be a batch mode session, exactly as if the **-s** option had been specified.

FAST STARTUP

This section will tell you the minimum amount that you need to do simple editing tasks using **vi**. If you've never used any screen editor before, you're likely to have problems even with this simple introduction. In that case you should find someone that already knows **vi** and have them walk you through this section.

vi is a screen editor. This means that it takes up almost the entire screen, displaying part of the file on each screen line, except for the last line of the screen. The last line of the screen is used for you to give commands to **vi**, and for **vi** to give information to you.

The other fact that you need to understand is that **vi** is a modeful editor, i.e. you are either entering text or you are executing commands, and you have to be in the right mode to do one or the other. You will be in command mode when you first start editing a file. There are commands that switch you into input mode. There is only one key that takes you out of input mode, and that is the <escape> key.

Key names are written using angle brackets, e.g. <escape> means the "escape" key, usually labeled "Esc" on your terminal's keyboard. If

you're ever confused as to which mode you're in, keep entering the <escape> key until **vi** beeps at you. Generally, **vi** will beep at you if you try and do something that's not allowed. It will also display error messages.

To start editing a file, enter the following command:

```
$ vi file
```

The command you should enter as soon as you start editing is:

```
:set verbose showmode
```

This will make the editor give you verbose error messages and display the current mode at the bottom of the screen.

The commands to move around the file are:

h Move the cursor left one character.

j Move the cursor down one line.

k Move the cursor up one line.

l Move the cursor right one character.

<cursor-arrows>
The cursor arrow keys should work, too.

/text Search for the string "text" in the file, and move the cursor to its first character.

The commands to enter new text are:

a Append new text, after the cursor.

i Insert new text, before the cursor.

O Open a new line above the line the cursor is on, and start entering text.

o Open a new line below the line the cursor is on, and start entering text.

<escape> Once you've entered input mode using one of the **a**, **i**, **O** or **o** commands, use **<escape>** to quit entering text and return to command mode.

The commands to copy text are:

p Append the copied line after the line the cursor is on.

yy Copy the line the cursor is on.

The commands to delete text are:

dd Delete the line the cursor is on.

x Delete the character the cursor is on.

The commands to write the file are:

:w Write the file back to the file with the name that you

originally used as an argument on the **vi** command line.

:w *file_name*
Write the file back to the file with the name *file_name*.

The commands to quit editing and exit the editor are:

:q Quit editing and leave **vi** (if you've modified the file, but not saved your changes, **vi** will refuse to quit).

:q! Quit, discarding any modifications that you may have made.

One final caution: Unusual characters can take up more than one column on the screen, and long lines can take up more than a single screen line. The above commands work on "physical" characters and lines, i.e. they affect the entire line no matter how many screen lines it takes up and the entire character no matter how many screen columns it takes up.

REGULAR EXPRESSIONS

ex/vi supports regular expressions (REs), as documented in re_format(7), for line addresses, as the first part of the **ex substitute**, **global** and **v** commands, and in search patterns. Basic regular expressions (BREs) are enabled by default; extended regular expressions (EREs) are used if the **extended** option is enabled. The use of regular expressions can be largely disabled using the **magic** option.

The following strings have special meanings in the **ex/vi** version of regular expressions:

- An empty regular expression is equivalent to the last regular expression used.

- '\<' matches the beginning of the word.

- '\>' matches the end of the word.

- '~' matches the replacement part of the last **substitute** command.

BUFFERS

A buffer is an area where commands can save changed or deleted text for later use. **vi** buffers are named with a single character preceded by a double quote, for example "<c>; **ex** buffers are the same, but without the double quote. **nex/nvi** permits the use of any character without another meaning in the position where a buffer name is expected.

All buffers are either in *line mode* or *character mode*. Inserting a buffer in line mode into the text creates new lines for each of the lines it contains, while a buffer in character mode creates new lines for any lines *other* than the first and last lines it contains. The first and last lines are inserted at the current cursor position, becoming part of the current line. If there is more than one line in the buffer, the current line itself will be split. All **ex** commands which store text into buffers do so in line mode. The behaviour of **vi** commands depend on their associated motion command:

- **<control-A>**, **h**, **l**, **,**, **0**, **B**, **E**, **F**, **T**, **W**, **^**, **b**, **e**, **f** and **t** make the destination buffer character-oriented.

- **j**, **<control-M>**, **k**, **'**, **-**, **G**, **H**, **L**, **M**, _ and **|** make the destination buffer line-oriented.

- **$**, **%**, **‘**, **(**, **)**, **/**, **?**, **[[**, **]]**, **{** and **}** make the destination buffer character-oriented, unless the starting and end positions are the first and last characters on a line. In that case, the buffer is line-oriented.

The **ex** command **display buffers** displays the current mode for each buffer.

Buffers named ‘a’ through ‘z’ may be referred to using their uppercase equivalent, in which case new content will be appended to the buffer, instead of replacing it.

Buffers named ‘1’ through ‘9’ are special. A region of text modified using the **c** (change) or **d** (delete) commands is placed into the numeric buffer ‘1’ if no other buffer is specified and if it meets one of the following conditions:

- It includes characters from more than one line.

- It is specified using a line-oriented motion.

- It is specified using one of the following motion commands: **<control-A>**, **‘<character>**, **n**, **N**, **%**, **/**, **{**, **}**, **(**, **)**, and **?**.

Before this copy is done, the previous contents of buffer ‘1’ are moved into buffer ‘2’, ‘2’ into buffer ‘3’, and so on. The contents of buffer ‘9’ are discarded. Note that this rotation occurs *regardless* of the user specifying another buffer. In **vi**, text may be explicitly stored into the numeric buffers. In this case, the buffer rotation occurs before the replacement of the buffer’s contents. The numeric buffers are only available in **vi** mode.

VI COMMANDS

The following section describes the commands available in the command mode of the **vi** editor. The following words have a special meaning in the commands description:

bigword A set of non-whitespace characters.

buffer Temporary area where commands may place text. If not specified, the default buffer is used. See also *BUFFERS*, above.

count A positive number used to specify the desired number of iterations of a command. It defaults to 1 if not specified.

motion A cursor movement command which indicates the other end of the affected region of text, the first being the current cursor position. Repeating the command character makes it affect the whole current line.

word A sequence of letters, digits or underscores.

buffer and *count*, if both present, may be specified in any order. *motion* and *count*, if both present, are effectively multiplied together and considered part of the motion.

<control-A>

Search forward for the word starting at the cursor position.

[*count*] **<control-B>**

Page backwards *count* screens. Two lines of overlap are maintained, if possible.

[*count*] **<control-D>**

Scroll forward *count* lines. If *count* is not given, scroll forward the number of lines specified by the last **<control-D>**

or **<control-U>** command. If this is the first **<control-D>** command, scroll half the number of lines in the current screen.

[*count*] **<control-E>**
Scroll forward *count* lines, leaving the current line and column as is, if possible.

[*count*] **<control-F>**
Page forward *count* screens. Two lines of overlap are maintained, if possible.

<control-G>
Display the following file information: the file name (as given to **vi**); whether the file has been modified since it was last written; if the file is readonly; the current line number; the total number of lines in the file; and the current line number as a percentage of the total lines in the file.

[*count*] **<control-H>**
[*count*] **h**
Move the cursor back *count* characters in the current line.

[*count*] **<control-J>**
[*count*] **<control-N>**
[*count*] **j**
Move the cursor down *count* lines without changing the current column.

<control-L>
<control-R>

Repaint the screen.

[*count*] **<control-M>**
[*count*] **+**
Move the cursor down *count* lines to the first non-blank character of that line.

[*count*] **<control-P>**
[*count*] **k**
Move the cursor up *count* lines, without changing the current column.

<control-T>
Return to the most recent tag context.

[*count*] **<control-U>**
Scroll backwards *count* lines. If *count* is not given, scroll backwards the number of lines specified by the last **<control-D>** or **<control-U>** command. If this is the first **<control-U>** command, scroll half the number of lines in the current screen.

<control-W>
Switch to the next lower screen in the window, or to the first screen if there are no lower screens in the window.

[*count*] **<control-Y>**
Scroll backwards *count* lines, leaving the current line and column as is, if possible.

<control-Z>

Suspend the current editor session.

<escape>

Execute the **ex** command being entered, or cancel it if it is only partial.

<control-]>

Push a tag reference onto the tag stack.

<control-^>

Switch to the most recently edited file.

[*count*] **<space>**
[*count*] **l**

Move the cursor forward *count* characters without changing the current line.

[*count*] **!** *motion shell-argument(s)* <carriage-return>

Replace the lines spanned by *count* and *motion* with the output (standard output and standard error) of the program named by the **shell** option, called with a **-c** flag followed by the *shell-argument(s)* (bundled into a single argument). Within *shell-argument(s)*, the '%', '#' and '!' characters are expanded to the current file name, the previous current file name, and the command text of the previous **!** or **:!** commands, respectively. The special meaning of '%', '#' and '!' can be overridden by escaping them with a backslash.

[*count*] **# #|+|-**

Increment (trailing '#' or '+') or decrement (trailing '-') the number under the cursor by *count*, starting at the cursor

position or at the first non-blank character following it. Numbers with a leading '0x' or '0X' are interpreted as hexadecimal numbers. Numbers with a leading '0' are interpreted as octal numbers unless they contain a non-octal digit. Other numbers may be prefixed with a '+' or '-' sign.

[*count*] **$**

Move the cursor to the end of a line. If *count* is specified, additionally move the cursor down *count* - 1 lines.

% Move to the parenthesis, square bracket or curly brace matching the one found at the cursor position or the closest to the right of it.

& Repeat the previous substitution command on the current line.

'<*character*>

`<*character*>

Return to the cursor position marked by the character *character*, or, if *character* is '' or ``, to the position of the cursor before the last of the following commands: **<control-A>**, **<control-T>**, **<control-]>**, **%**, **'**, **`**, **(**, **)**, **/**, **?**, **G**, **H**, **L**, **[[**, **]]**, **{**, **}**. The first form returns to the first non-blank character of the line marked by *character*. The second form returns to the line and column marked by *character*.

[*count*] **(**

[*count*] **)**

Move *count* sentences backward or forward, respectively. A sentence is an area of text that begins with the first nonblank character following the previous sentence, paragraph, or

section boundary and continues until the next period, exclamation mark, or question mark character, followed by any number of closing parentheses, brackets, double or single quote characters, followed by either an end-of-line or two whitespace characters. Groups of empty lines (or lines containing only whitespace characters) are treated as a single sentence.

[*count*] **,**

Reverse find character (i.e. the last **F**, **f**, **T** or **t** command) *count* times.

[*count*] **-**

Move to the first non-blank character of the previous line, *count* times.

[*count*] **.**

Repeat the last **vi** command that modified text. *count* replaces both the *count* argument of the repeated command and that of the associated *motion*. If the **.** command repeats the **u** command, the change log is rolled forward or backward, depending on the action of the **u** command.

/*RE* <carriage-return>
/*RE***/** [*offset*] [**z**] <carriage-return>
?*RE* <carriage-return>
?*RE*? [*offset*] [**z**] <carriage-return>
N
n Search forward (‘/’) or backward (‘?’) for a regular expression. **n** and **N** repeat the last search in the same or opposite directions, respectively. If *RE* is empty, the last

search regular expression is used. If *offset* is specified, the cursor is placed *offset* lines before or after the matched regular expression. If either **n** or **N** commands are used as motion components for the **!** command, there will be no prompt for the text of the command and the previous **!** will be executed. Multiple search patterns may be grouped together by delimiting them with semicolons and zero or more whitespace characters. These patterns are evaluated from left to right with the final cursor position determined by the last search pattern. A **z** command may be appended to the closed search expressions to reposition the result line.

0 Move to the first character in the current line.

: Execute an **ex** command.

[*count*] **;**
Repeat the last character find (i.e. the last **F**, **f**, **T** or **t** command) *count* times.

[*count*] **<** *motion*
[*count*] **>** *motion*
Shift *count* lines left or right, respectively, by an amount of **shiftwidth**.

@ *buffer*
Execute a named *buffer* as **vi** commands. The buffer may include **ex** commands too, but they must be expressed as a **:** command. If *buffer* is '@' or '*', then the last buffer executed shall be used.

[*count*] **A**

Enter input mode, appending the text after the end of the line. If a *count* argument is given, the characters input are repeated *count* - 1 times after input mode is exited.

[*count*] **B**

Move backwards *count* bigwords.

[*buffer*] **C**

Change text from the current position to the end-of-line. If *buffer* is specified, "yank" the deleted text into *buffer*.

[*buffer*] **D**

Delete text from the current position to the end-of-line. If *buffer* is specified, "yank" the deleted text into *buffer*.

[*count*] **E**

Move forward *count* end-of-bigwords.

[*count*] **F** <*character*>

Search *count* times backward through the current line for *character*.

[*count*] **G**

Move to line *count*, or the last line of the file if *count* is not specified.

[*count*] **H**

Move to the screen line *count* - 1 lines below the top of the screen.

[*count*] **I**

Enter input mode, inserting the text at the beginning of the line. If a *count* argument is given, the characters input are repeated *count* - 1 more times.

[*count*] **J**

Join *count* lines with the current line. The spacing between two joined lines is set to two whitespace characters if the former ends with a question mark, a period or an exclamation mark. It is set to one whitespace character otherwise.

[*count*] **L**

Move to the screen line *count* - 1 lines above the bottom of the screen.

M Move to the screen line in the middle of the screen.

[*count*] **O**

Enter input mode, appending text in a new line above the current line. If a *count* argument is given, the characters input are repeated *count* - 1 more times.

[*buffer*] **P**

Insert text from *buffer* before the current column if *buffer* is character-oriented or before the current line if it is line-oriented.

Q Exit **vi** (or visual) mode and switch to **ex** mode.

[*count*] **R**

Enter input mode, replacing the characters in the current line.

If a *count* argument is given, the characters input are repeated *count* - 1 more times upon exit from insert mode.

[*buffer*] [*count*] **S**
Substitute *count* lines. If *buffer* is specified, "yank" the deleted text into *buffer*.

[*count*] **T** <*character*>
Search backwards, *count* times, through the current line for the character after the specified *character*.

U Restore the current line to its state before the cursor last moved to it.

[*count*] **W**
Move forward *count* bigwords.

[*buffer*] [*count*] **X**
Delete *count* characters before the cursor, on the current line. If *buffer* is specified, "yank" the deleted text into *buffer*.

[*buffer*] [*count*] **Y**
Copy (or "yank") *count* lines into *buffer*.

ZZ Write the file and exit **vi** if there are no more files to edit. Entering two "quit" commands in a row ignores any remaining file to edit.

[*count*] **[[**
Back up *count* section boundaries.

[*count*] **]]**
Move forward *count* section boundaries.

^ Move to the first non-blank character on the current line.

[*count*] **_**
Move down *count* - 1 lines, to the first non-blank character.

[*count*] **a**
Enter input mode, appending the text after the cursor. If a *count* argument is given, the characters input are repeated *count* -1 more times.

[*count*] **b**
Move backwards *count* words.

[*buffer*] [*count*] **c** *motion*
Change the region of text described by *count* and *motion*. If *buffer* is specified, "yank" the changed text into *buffer*.

[*buffer*] [*count*] **d** *motion*
Delete the region of text described by *count* and *motion*. If *buffer* is specified, "yank" the deleted text into *buffer*.

[*count*] **e**
Move forward *count* end-of-words.

[*count*] **f** <*character*>
Search forward, *count* times, through the rest of the current line for <*character*>.

[*count*] **i**

Enter input mode, inserting the text before the cursor. If a *count* argument is given, the characters input are repeated *count* -1 more times.

m <*character*>

Save the current context (line and column) as <*character*>.

[*count*] **o**

Enter input mode, appending text in a new line under the current line. If a *count* argument is given, the characters input are repeated *count* - 1 more times.

[*buffer*] **p**

Append text from *buffer*. Text is appended after the current column if *buffer* is character oriented, or the after current line otherwise.

[*count*] **r** <*character*>

Replace *count* characters by *character*.

[*buffer*] [*count*] **s**

Substitute *count* characters in the current line starting with the current character. If *buffer* is specified, "yank" the substituted text into *buffer*.

[*count*] **t** <*character*>

Search forward, *count* times, through the current line for the character immediately before <*character*>.

u Undo the last change made to the file. If repeated, the **u**

command alternates between these two states. The **.** command, when used immediately after **u**, causes the change log to be rolled forward or backward, depending on the action of the **u** command.

[*count*] **w**

Move forward *count* words.

[*buffer*] [*count*] **x**

Delete *count* characters at the current cursor position, but no more than there are till the end of the line.

[*buffer*] [*count*] **y** *motion*

Copy (or "yank") a text region specified by *count* and *motion* into a buffer.

[*count1*] **z** [*count2*] **type**

Redraw, optionally repositioning and resizing the screen. If *count2* is specified, limit the screen size to *count2* lines. The following **type** characters may be used:

+ If *count1* is specified, place the line *count1* at the top of the screen. Otherwise, display the screen after the current screen.

<carriage-return>

Place the line *count1* at the top of the screen.

. Place the line *count1* in the center of the screen.

- Place the line *count1* at the bottom of the screen.

^ If *count1* is given, display the screen before the screen before *count1* (i.e. 2 screens before). Otherwise, display the screen before the current screen.

[*count*] **{**
Move backward *count* paragraphs.

[*column*] **|**
Move to a specific *column* position on the current line. If *column* is omitted, move to the start of the current line.

[*count*] **}**
Move forward *count* paragraphs.

[*count*] **~** *motion*
If the **tildeop** option is not set, reverse the case of the next *count* character(s) and no *motion* can be specified. Otherwise *motion* is mandatory and **~** reverses the case of the characters in a text region specified by the *count* and *motion*.

<interrupt>
Interrupt the current operation. The <interrupt> character is usually <control-C>.

VI TEXT INPUT COMMANDS

The following section describes the commands available in the text input mode of the **vi** editor.

<nul> Replay the previous input.

<control-D>

Erase to the previous *shiftwidth* column boundary.

^<control-D>

Erase all of the autoindent characters, and reset the autoindent level.

0<control-D>

Erase all of the autoindent characters.

<control-T>

Insert sufficient <tab> and <space> characters to move forward to the next *shiftwidth* column boundary.

<erase>
<control-H>

Erase the last character.

<literal next>

Escape the next character from any special meaning. The <literal next> character is usually <control-V>.

<escape>

Resolve all text input into the file, and return to command mode.

<line erase>

Erase the current line.

<control-W>
<word erase>

Erase the last word. The definition of word is dependent on the **altwerase** and **ttywerase** options.

<control-X>[0-9A-Fa-f]+

Insert a character with the specified hexadecimal value into the text.

<interrupt>

Interrupt text input mode, returning to command mode. The <interrupt> character is usually <control-C>.

EX COMMANDS

The following section describes the commands available in the **ex** editor. In each entry below, the tag line is a usage synopsis for the command.

<end-of-file>

Scroll the screen.

! *argument(s)*
[*range*] **!** *argument(s)*

Execute a shell command, or filter lines through a shell command.

" A comment.

[*range*] **nu**[**mber**] [*count*] [*flags*]
[*range*] **#** [*count*] [*flags*]

Display the selected lines, each preceded with its line number.

@ buffer
***** *buffer*
Execute a buffer.

[*range*] **<**[**< ...**] [*count*] [*flags*]
Shift lines left.

[*line*] **=** [*flags*]
Display the line number of *line*. If *line* is not specified, display the line number of the last line in the file.

[*range*] **>**[**> ...**] [*count*] [*flags*]
Shift lines right.

ab[**breviate**] *lhs rhs*
vi only. Add *lhs* as an abbreviation for *rhs* to the abbreviation list.

[*line*] **a**[**ppend**][**!**]
The input text is appended after the specified line.

ar[**gs**] Display the argument list.

bg **vi** only. Background the current screen.

[*range*] **c**[**hange**][**!**] [*count*]
The input text replaces the specified range.

chd[**ir**][**!**] [*directory*]
cd[**!**] [*directory*]
Change the current working directory.

[*range*] **co**[**py**] *line* [*flags*]
[*range*] **t** *line* [*flags*]
Copy the specified lines after the destination *line*.

cs[**cope**] **add | find | help | kill | reset**
Execute a Cscope command.

[*range*] **d**[**elete**] [*buffer*] [*count*] [*flags*]
Delete the lines from the file.

di[**splay**] **b**[**uffers**] | **c**[**onnections**] | **s**[**creens**] | **t**[**ags**]
Display buffers, Cscope connections, screens or tags.

[**Ee**][**dit**][**!**] [+*cmd*] [*file*]
[**Ee**]**x**[**!**] [+*cmd*] [*file*]
Edit a different file.

exu[**sage**] [*command*]
Display usage for an **ex** command.

f[**ile**] [*file*]
Display and optionally change the file name.

[**Ff**]**g** [*name*]
vi mode only. Foreground the specified screen.

[*range*] **g**[**lobal**] /*pattern*/ [*commands*]
[*range*] **v** /*pattern*/ [*commands*]
Apply commands to lines matching (‘global’) or not matching (‘v’) a pattern.

he[**lp**] Display a help message.

[*line*] **i**[**nsert**][**!**]
The input text is inserted before the specified line.

[*range*] **j**[**oin**][**!**] [*count*] [*flags*]
Join lines of text together.

[*range*] **l**[**ist**] [*count*] [*flags*]
Display the lines unambiguously.

map[**!**] [*lhs rhs*]
Define or display maps (for **vi** only).

[*line*] **ma**[**rk**] <*character*>
[*line*] **k** <*character*>
Mark the line with the mark <*character*>.

[*range*] **m**[**ove**] *line*
Move the specified lines after the target line.

mk[**exrc**][**!**] *file*
Write the abbreviations, editor options and maps to the specified *file*.

[**Nn**][**ext**][**!**] [*file ...*]
Edit the next file from the argument list.

pre[**serve**]
Save the file in a form that can later be recovered using the **ex -r** option.

[**Pp**]**rev**[**ious**][**!**]
Edit the previous file from the argument list.

[*range*] **p**[**rint**] [*count*] [*flags*]
Display the specified lines.

[*line*] **pu**[**t**] [*buffer*]
Append buffer contents to the current line.

q[**uit**][**!**]
End the editing session.

[*line*] **r**[**ead**][**!**] [*file*]
Read a file.

rec[**over**] *file*
Recover *file* if it was previously saved.

res[**ize**] [**+**|**-**]*size*
vi mode only. Grow or shrink the current screen.

rew[**ind**][**!**]
Rewind the argument list.

se[**t**] [option[=[value]] ...] [nooption ...] [option? ...] [*all*]
Display or set editor options.

sh[**ell**] Run a shell program.

so[**urce**] *file*
Read and execute **ex** commands from a file.

[*range*] **s**[**ubstitute**] [/*pattern*/*replace*/] [*options*] [*count*] [*flags*]
[*range*] **&** [*options*] [*count*] [*flags*]
[*range*] **~** [*options*] [*count*] [*flags*]

Make substitutions. The *replace* field may contain any of the following sequences:

‘&’ The text matched by *pattern*.

‘~’ The replacement part of the previous **substitute** command.

‘%’ If this is the entire *replace* pattern, the replacement part of the previous **substitute** command.

‘\#’ Where ‘#’ is an integer from 1 to 9, the text matched by the #’th subexpression in *pattern*.

‘\L’ Causes the characters up to the end of the line of the next occurrence of ‘\E’ or ‘\e’ to be converted to lowercase.

‘\l’ Causes the next character to be converted to lowercase.

‘\U’ Causes the characters up to the end of the line of the next occurrence of ‘\E’ or ‘\e’ to be converted to uppercase.

‘\u’ Causes the next character to be converted to uppercase.

su[**spend**][**!**]
st[**op**][**!**]
<suspend>
Suspend the edit session. The <suspend> character is usually <control-Z>.

[**Tt**]**a**[**g**][**!**] *tagstring*
Edit the file containing the specified tag.

tagn[**ext**][**!**]
Edit the file containing the next context for the current tag.

tagp[**op**][**!**] [*file* | *number*]
Pop to the specified tag in the tags stack.

tagpr[**ev**][**!**]
Edit the file containing the previous context for the current tag.

tagt[**op**][**!**]
Pop to the least recent tag on the tags stack, clearing the stack.

una[**bbreviate**] *lhs*
vi only. Delete an abbreviation.

u[**ndo**] Undo the last change made to the file.

unm[**ap**][**!**] *lhs*
Unmap a mapped string.

ve[**rsion**]
Display the version of the **ex/vi** editor.

[*line*] **vi**[**sual**] [*type*] [*count*] [*flags*]
ex mode only. Enter **vi**.

[**Vi**]**i**[**sual**][**!**] [+*cmd*] [*file*]
vi mode only. Edit a new file.

viu[**sage**] [*command*]
Display usage for a **vi** command.

[*range*] **w**[**rite**][**!**] [>>] [*file*]
[*range*] **w**[**rite**] [**!**] [*file*]
[*range*] **wn**[**!**] [>>] [*file*]
[*range*] **wq**[**!**] [>>] [*file*]
Write the file.

[*range*] **x**[**it**][**!**] [*file*]
Exit the editor, writing the file if it has been modified.

[*range*] **ya**[**nk**] [*buffer*] [*count*]
Copy the specified lines to a buffer.

[*line*] **z** [*type*] [*count*] [*flags*]
Adjust the window.

SET OPTIONS

There are a large number of options that may be set (or unset) to change the editor's behavior. This section describes the options, their abbreviations and their default values.

In each entry below, the first part of the tag line is the full name of the option, followed by any equivalent abbreviations. The part in square brackets is the default value of the option. Most of the options are boolean, i.e. they are either on or off, and do not have an associated value.

Options apply to both **ex** and **vi** modes, unless otherwise specified.

altwerase [off]
vi only. Select an alternate word erase algorithm.

autoindent, **ai** [off]
Automatically indent new lines.

autoprint, **ap** [on]
ex only. Display the current line automatically.

autowrite, **aw** [off]
Write modified files automatically when changing files or suspending the editor session.

backup [""]
Back up files before they are overwritten.

beautify, **bf** [off]
Discard control characters.

cdpath [environment variable CDPATH, or current directory]
The directory paths used as path prefixes for the **cd** command.

cedit [no default]
Set the character to edit the colon command-line history.

columns, **co** [80]
Set the number of columns in the screen.

comment [off]
vi only. Skip leading comments in shell, C and C++ language files.

directory, **dir** [environment variable TMPDIR, or /tmp]
The directory where temporary files are created.

edcompatible, **ed** [off]
Remember the values of the 'c' and 'g' suffixes to the **substitute** commands, instead of initializing them as unset for each new command.

escapetime [1]
The 10th's of a second **ex/vi** waits for a subsequent key to complete an <escape> key mapping.

errorbells, **eb** [off]
ex only. Announce error messages with a bell.

exrc, **ex** [off]
Read the startup files in the local directory.

extended [off]
Use extended regular expressions (EREs) rather than basic regular expressions (BREs). See re_format(7) for more

information on regular expressions.

filec [no default]
Set the character to perform file path completion on the colon command line.

flash [off]
Flash the screen instead of beeping the keyboard on error.

hardtabs, **ht** [0]
Set the spacing between hardware tab settings. This option currently has no effect.

iclower [off]
Makes all regular expressions case-insensitive, as long as an upper-case letter does not appear in the search string.

ignorecase, **ic** [off]
Ignore case differences in regular expressions.

keytime [6]
The 10th's of a second **ex/vi** waits for a subsequent key to complete a key mapping.

leftright [off]
vi only. Do left-right scrolling.

lines, **li** [24]
vi only. Set the number of lines in the screen.

lisp [off]

vi only. Modify various search commands and options to work with Lisp. This option is not yet implemented.

list [off]

Display lines in an unambiguous fashion.

lock [on]

Attempt to get an exclusive lock on any file being edited, read or written.

magic [on]

When turned off, all regular expression characters except for '^' and '$' are treated as ordinary characters. Preceding individual characters by '\' re-enables them.

matchtime [7]

vi only. The 10th's of a second **ex/vi** pauses on the matching character when the **showmatch** option is set.

mesg [on]

Permit messages from other users.

msgcat [/usr/share/vi/catalog/]

Selects a message catalog to be used to display error and informational messages in a specified language.

modelines, **modeline** [off]

Read the first and last few lines of each file for **ex** commands. This option will never be implemented.

noprint [""]

Characters that are never handled as printable characters.

number, nu [off]
Precede each line displayed with its current line number.

octal [off]
Display unknown characters as octal numbers, instead of the default hexadecimal.

open [on]
ex only. If this option is not set, the **open** and **visual** commands are disallowed.

optimize, **opt** [on]
vi only. Optimize text throughput to dumb terminals. This option is not yet implemented

paragraphs, **para** [IPLPPPQPP LIpplpipbp]
vi only. Define additional paragraph boundaries for the **{** and **}** commands.

path [""]
Define additional directories to search for files being edited.

print [""]
Characters that are always handled as printable characters.

prompt [on]
ex only. Display a command prompt.

readonly, **ro** [off]

Mark the file and session as read-only.

recdir [/var/tmp/vi.recover]
The directory where recovery files are stored.

redraw, **re** [off]
vi only. Simulate an intelligent terminal on a dumb one. This option is not yet implemented.

remap [on]
Remap keys until resolved.

report [5]
Set the number of lines about which the editor reports changes or yanks.

ruler [off]
vi only. Display a row/column ruler on the colon command line.

scroll, **scr** [($LINES - 1) / 2]
Set the number of lines scrolled.

searchincr [off]
Makes the **/** and **?** commands incremental.

sections, **sect** [NHSHH HUnhsh]
vi only. Define additional section boundaries for the **[[** and **]]** commands.

secure [off]

Turns off all access to external programs.

shell, **sh** [environment variable SHELL, or /bin/sh]
Select the shell used by the editor.

shellmeta [~{[*?$‘’"\]
Set the meta characters checked to determine if file name expansion is necessary.

shiftwidth, **sw** [8]
Set the autoindent and shift command indentation width.

showmatch, **sm** [off]
vi only. Note matching ‘{’ and ‘(’ for ‘}’ and ‘)’ characters.

showmode, **smd** [off]
vi only. Display the current editor mode and a "modified" flag.

sidescroll [16]
vi only. Set the amount a left-right scroll will shift.

slowopen, **slow** [off]
Delay display updating during text input. This option is not yet implemented.

sourceany [off]
Read startup files not owned by the current user. This option will never be implemented.

tabstop, **ts** [8]

This option sets tab widths for the editor display.

taglength, **tl** [0]
Set the number of significant characters in tag names.

tags, tag [tags]
Set the list of tags files.

term, **ttytype**, **tty** [environment variable TERM]
Set the terminal type.

terse [off]
This option has historically made editor messages less verbose. It has no effect in this implementation.

tildeop [off]
Modify the ~ command to take an associated motion.

timeout, **to** [on]
Time out on keys which may be mapped.

ttywerase [off]
vi only. Select an alternate erase algorithm.

verbose [off]
vi only. Display an error message for every error.

w300 [no default]
vi only. Set the window size if the baud rate is less than 1200 baud.

w1200 [no default]
vi only. Set the window size if the baud rate is equal to 1200 baud.

w9600 [no default]
vi only. Set the window size if the baud rate is greater than 1200 baud.

warn [on]
ex only. This option causes a warning message to be printed on the terminal if the file has been modified since it was last written, before a **!** command.

window, **w**, **wi** [environment variable LINES - 1]
Set the window size for the screen.

windowname [off]
Change the icon/window name to the current file name even if it can't be restored on editor exit.

wraplen, **wl** [0]
vi only. Break lines automatically, the specified number of columns from the left-hand margin. If both the **wraplen** and **wrapmargin** edit options are set, the **wrapmargin** value is used.

wrapmargin, **wm** [0]
vi only. Break lines automatically, the specified number of columns from the right-hand margin. If both the **wraplen** and **wrapmargin** edit options are set, the **wrapmargin** value is used.

wrapscan, **ws** [on]
Set searches to wrap around the end or beginning of the file.

writeany, **wa** [off]
Turn off file-overwriting checks.

ENVIRONMENT

COLUMNS The number of columns on the screen. This value overrides any system or terminal specific values. If the COLUMNS environment variable is not set when **ex/vi** runs, or the **columns** option is explicitly reset by the user, **ex/vi** enters the value into the environment.

EXINIT A list of **ex** startup commands, read after */etc/vi.exrc* unless the variable NEXINIT is also set.

HOME The user's home directory, used as the initial directory path for the startup *$HOME/.nexrc* and *$HOME/.exrc* files. This value is also used as the default directory for the **vi cd** command.

LINES The number of rows on the screen. This value overrides any system or terminal specific values. If the LINES environment variable is not set when **ex/vi** runs, or the **lines** option is explicitly reset by the user, **ex/vi** enters the value into the environment.

NEXINIT A list of **ex** startup commands, read after */etc/vi.exrc*.

SHELL The user's shell of choice (see also the **shell** option).

TERM The user's terminal type. The default is the type "unknown". If the TERM environment variable is not set when **ex/vi** runs, or the **term** option is explicitly reset by the user, **ex/vi** enters the value into the environment.

TMPDIR The location used to stored temporary files (see also the **directory** edit option).

ASYNCHRONOUS EVENTS

SIGALRM **vi/ex** uses this signal for periodic backups of file modifications and to display "busy" messages when operations are likely to take a long time.

SIGHUP
SIGTERM If the current buffer has changed since it was last written in its entirety, the editor attempts to save the modified file so it can be later recovered. See the **vi/ex** reference manual section *Recovery* for more information.

SIGINT When an interrupt occurs, the current operation is halted and the editor returns to the command level. If interrupted during text input, the text already input is resolved into the file as if the text input had been normally terminated.

SIGWINCH The screen is resized. See the **vi/ex** reference manual section *Sizing the Screen* for more information.

FILES

/bin/sh The default user shell.

/etc/vi.exrc System-wide **vi** startup file. It is read for **ex** commands first in the startup sequence. Must be owned by root or the user, and writable only by the owner.

/tmp Temporary file directory.

/var/tmp/vi.recover The default recovery file directory.

$HOME/.nexrc First choice for user's home directory startup file, read for **ex** commands right after */etc/vi.exrc* unless either NEXINIT or EXINIT are set. Must be owned by root or the user, and writable only by the owner.

$HOME/.exrc Second choice for user's home directory startup file, read for **ex** commands under the same conditions as *$HOME/.nexrc*.

.nexrc First choice for local directory startup file, read for **ex** commands at the end of the startup sequence if the **exrc** option was turned on earlier. Must be owned by the user and writable only by the owner.

.exrc Second choice for local directory startup file, read for **ex** commands under the same conditions as *.nexrc*.

EXIT STATUS

The **ex** and **vi** utilities exit 0 on success, and >0 if an error occurs.

SEE ALSO

ctags(1), re_format(7)

HISTORY

An **ex** command appeared in 1BSD. The **nex/nvi** replacements for the **ex/vi** editor appeared in 4.4BSD.

NAME

wc - word, line, and byte or character count

SYNOPSIS

wc [**-c** | **-m**] [**-hlw**] [*file* ...]

DESCRIPTION

The **wc** utility reads one or more input text files and, by default, writes the number of lines, words, and bytes contained in each input file to the standard output. If more than one input file is specified, a line of cumulative count(s) for all named files is output on a separate line following the last file count. **wc** considers a word to be a maximal string of characters delimited by whitespace. Whitespace characters are the set of characters for which the isspace(3) function returns true.

The options are as follows:

-c The number of bytes in each input file is written to the standard output.

-h Use unit suffixes: Byte, Kilobyte, Megabyte, Gigabyte, Terabyte, Petabyte, and Exabyte in order to reduce the number of digits to four or fewer using powers of 2 for sizes (K=1024, M=1048576, etc.).

-l The number of lines in each input file is written to the standard output.

-m The number of characters in each input file is written to the standard output.

-w The number of words in each input file is written to the standard output.

When an option is specified, **wc** only reports the information requested by that option. The default action is equivalent to the flags **-clw** having been specified. The **-c** and **-m** options are mutually exclusive.

If no file names are specified, the standard input is used and a file name is not output. The resulting output is one line of the requested count(s) with the cumulative sum of all files read in via standard input.

By default, the standard output contains a line for each input file of the form:

lines words bytes file_name

If the **-m** option is specified, the number of bytes is replaced by the number of characters in the listing above. The counts for lines, words, and bytes (or characters) are integers separated by spaces.

EXIT STATUS

The **wc** utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

isspace(3)

HISTORY

A **wc** command appeared in Version 1 AT&T UNIX.

NAME

xargs - construct argument list(s) and execute utility

SYNOPSIS

xargs [**-0oprt**] [**-E** *eofstr*] [**-I** *replstr* [**-R** *replacements*]] [**-J** *replstr*] [**-L** *number*] [**-n** *number* [**-x**]] [**-P** *maxprocs*] [**-s** *size*] [*utility* [*argument ...*]]

DESCRIPTION

The **xargs** utility reads space, tab, newline, and end-of-file delimited strings from the standard input and executes the specified *utility* with the strings as arguments.

Any arguments specified on the command line are given to the *utility* upon each invocation, followed by some number of the arguments read from standard input. The *utility* is repeatedly executed one or more times until standard input is exhausted.

Spaces, tabs and newlines may be embedded in arguments using single (‘’’) or double (‘"’) quotes or backslashes (‘\’). Single quotes escape all non-single quote characters, excluding newlines, up to the matching single quote. Double quotes escape all non-double quote characters, excluding newlines, up to the matching double quote. Any single character, including newlines, may be escaped by a backslash.

The options are as follows:

-0 Change **xargs** to expect NUL (‘\0’) characters as separators, instead of spaces and newlines. The quoting mechanisms described above are not performed. This option is expected to be used in concert with the **-print0** function in find(1).

-E *eofstr*

Use *eofstr* as a logical EOF marker.

-I *replstr*

Execute *utility* for each input line, replacing one or more occurrences of *replstr* in up to *replacements* (or 5 if no **-R** flag is specified) arguments to *utility* with the entire line of input. The resulting arguments, after replacement is done, will not be allowed to grow beyond 255 bytes; this is implemented by concatenating as much of the argument containing *replstr* as possible, to the constructed arguments to *utility*, up to 255 bytes. The 255 byte limit does not apply to arguments to *utility* which do not contain *replstr*, and furthermore, no replacement will be done on *utility* itself. Implies **-x**.

-J *replstr*

If this option is specified, **xargs** will use the data read from standard input to replace the first occurrence of *replstr* instead of appending that data after all other arguments. This option will not effect how many arguments will be read from input (**-n**), or the size of the command(s) **xargs** will generate (**-s**). The option just moves where those arguments will be placed in the command(s) that are executed. The *replstr* must show up as a distinct *argument* to **xargs**. It will not be recognized if, for instance, it is in the middle of a quoted string. Furthermore, only the first occurrence of the *replstr* will be replaced. For example, the following command will copy the list of files and directories which start with an uppercase letter in the current directory to *destdir*:

/bin/ls -1d [A-Z]* | xargs -J % cp -Rp % destdir

-L *number*

Call *utility* for every *number* of non-empty lines read. A line ending in unescaped white space and the next non-empty line are considered to form one single line. If EOF is reached and fewer than *number* lines have been read then *utility* will be called with the available lines.

-n *number*

Set the maximum number of arguments taken from standard input for each invocation of *utility*. An invocation of *utility* will use less than *number* standard input arguments if the number of bytes accumulated (see the **-s** option) exceeds the specified *size* or there are fewer than *number* arguments remaining for the last invocation of *utility*. The current default value for *number* is 5000.

-o Reopen stdin as */dev/tty* in the child process before executing the command. This is useful if you want **xargs** to run an interactive application.

-P *maxprocs*

Parallel mode: run at most *maxprocs* invocations of *utility* at once.

-p Echo each command to be executed and ask the user whether it should be executed. An affirmative response, ‘y’ in the POSIX locale, causes the command to be executed, any other response causes it to be skipped. No commands are executed if the process is not attached to a terminal.

-R *replacements*

Specify the maximum number of arguments that **-I** will do replacement in. If *replacements* is negative, the number of arguments in which to replace is unbounded.

-r Do not run the command if there are no arguments. Normally the command is executed at least once even if there are no arguments.

-s *size* Set the maximum number of bytes for the command line length provided to *utility*. The sum of the length of the utility name, the arguments passed to *utility* (including NUL terminators) and the current environment will be less than or equal to this number. The current default value for *size* is ARG_MAX - 4096.

-t Echo the command to be executed to standard error immediately before it is executed.

-x Force **xargs** to terminate immediately if a command line containing *number* arguments will not fit in the specified (or default) command line length.

If no *utility* is specified, echo(1) is used.

Undefined behavior may occur if *utility* reads from the standard input.

The **xargs** utility exits immediately (without processing any further input) if a command line cannot be assembled, *utility* cannot be invoked, an invocation of *utility* is terminated by a signal, or an invocation of *utility* exits with a value of 255.

EXIT STATUS

xargs exits with one of the following values:

0	All invocations of *utility* returned a zero exit status.
123	One or more invocations of *utility* returned a nonzero exit status.
124	The *utility* exited with a 255 exit status.
125	The *utility* was killed or stopped by a signal.
126	The *utility* was found but could not be executed.
127	The *utility* could not be found.
1	Some other error occurred.

SEE ALSO

echo(1), find(1), execvp(3)

HISTORY

An **xargs** command appeared in PWB UNIX.

BUGS

If *utility* attempts to invoke another command such that the number of arguments or the size of the environment is increased, it risks execvp(3) failing with E2BIG.

NAME

yes - be repetitively affirmative

SYNOPSIS

yes [*expletive*]

DESCRIPTION

yes outputs *expletive*, or, by default, "y", forever.

HISTORY

A **yes** command appeared in Version 7 AT&T UNIX.

NAME

compress, **uncompress**, **zcat** - compress and expand data

SYNOPSIS

compress [**-cfrv**] [*file ...*]
uncompress [**-cfrv**] [*file ...*]
zcat [**-fr**] [*file ...*]

DESCRIPTION

The **compress** utility reduces the size of the named files using adaptive Lempel-Ziv coding, in compress mode. Each file is renamed to the same name plus the extension ".Z". As many of the modification time, access time, file flags, file mode, user ID, and group ID as allowed by permissions are retained in the new file. If compression would not reduce the size of a file, the file is ignored (unless **-f** is used).

The **uncompress** utility restores compressed files to their original form, renaming the files by removing the extension. It has the ability to restore files compressed by both **compress** and gzip(1), recognising the following extensions: ".Z", "-Z", "_Z", ".gz", "-gz", "_gz", ".tgz", "-tgz", "_tgz", ".taz", "-taz", and "_taz". Extensions ending in "tgz" and "taz" are not removed when decompressing, instead they are converted to "tar".

The **zcat** command is equivalent in functionality to **uncompress -c**.

If renaming the files would cause files to be overwritten and the standard input device is a terminal, the user is prompted (on the standard error output) for confirmation. If prompting is not possible or confirmation is not received, the files are not overwritten.

If no files are specified, the standard input is compressed or uncompressed to the standard output. If either the input or output files are not regular files, the checks for reduction in size and file overwriting are not performed, the input file is not removed, and the attributes of the input file are not retained.

The uncompressed file inherits the time stamp of the compressed version and the uncompressed file name is generated from the name of the compressed file as described above.

The options are as follows:

-c Compressed or uncompressed output is written to the standard output. No files are modified (force **zcat** mode).

-f Force compression of *file*, even if it is not actually reduced in size. Additionally, files are overwritten without prompting for confirmation. If the input data is not in a format recognized by **compress** and if the option **-c** is also given, copy the input data without change to the standard output: let **zcat** behave as cat(1).

-r Descend into specified directories recursively.

-v Print the percentage reduction of each file and other information.

compress uses a modified Lempel-Ziv algorithm (LZW). Common substrings in the file are first replaced by 9-bit codes 257 and up. When code 512 is reached, the algorithm switches to 10-bit codes and continues to use more bits until the limit of 16-bit codes is reached.

After the limit is reached, **compress** periodically checks the compression ratio. If it is increasing, **compress** continues to use the existing code dictionary. However, if the compression ratio decreases, **compress** discards the table of substrings and rebuilds it from scratch. This allows the algorithm to adapt to the next "block" of the file.

The amount of compression obtained depends on the size of the input and the distribution of common substrings. Typically, text such as source code or English is reduced by 50 - 60% using **compress**. Compression is generally much better than that achieved by Huffman coding, or adaptive Huffman coding, and takes less time to compute.

EXIT STATUS

The **compress** utility exits with one of the following values:

0	Success.
1	An error occurred.
2	At least one of the specified files was not compressed since **-f** was not specified and compression would have resulted in a size increase.
>2	An error occurred.

The **uncompress** and **zcat** utilities exit 0 on success, and >0 if an error occurs.

SEE ALSO

gzexe(1), gzip(1), zdiff(1), zforce(1), zmore(1), znew(1), compress(3)

HISTORY

A **compress** command appeared in 4.3BSD.

NAME

hier - layout of filesystems

DESCRIPTION

A sketch of the filesystem hierarchy.

/ Root directory.

/bin/ User utilities fundamental to both single and multi-user environments. These programs are statically compiled and therefore do not depend on any system libraries to run.

/dev/ Block and character device files.

/etc/ System configuration files and scripts.

/home/ Default location for user home directories.

/mnt/ Empty directory commonly used by system administrators as a temporary mount point.

/root/ Default home directory for the superuser.

/sbin/ System programs and administration utilities fundamental to both single and multi-user environments. These programs are statically compiled and therefore do not depend on any system libraries to run.

/tmp/ Temporary files that are *not* preserved between system reboots. Periodically cleaned by daily(8).

/usr/ Contains the majority of user utilities and applications.

- X11R6/ Files required for the X11 window system.
- bin/ Common utilities, programming tools, and applications.
- games/ Useful and semi-frivolous programs.
- include/ Standard C include files.
- lib/ System libraries.
- libdata/ Miscellaneous utility data files.
- libexec/ System daemons and utilities (executed by other programs).
- local/ Local executables, libraries, etc.
- sbin/ System daemons and utilities (executed by users).
- share/ Architecture independent data files.

/var/ Multi-purpose log, temporary, transient, and spool files.

SEE ALSO

apropos(1), find(1), locate(1), whatis(1), whereis(1), which(1)

HISTORY

A **hier** manual appeared in Version 7 AT&T UNIX.

NAME

glob - shell-style pattern matching

DESCRIPTION

Globbing characters (wildcards) are special characters used to perform pattern matching of pathnames and command arguments in the shell, as well as in the C library functions fnmatch(3) and glob(3).

A glob pattern is a word containing one or more unquoted ‘?’ or ‘*’ characters, or "[..]" sequences.

Globs should not be confused with the more powerful regular expressions used by programs such as awk(1), grep(1), and sed(1). While there is some overlap in the special characters used in regular expressions and globs, their meaning is different.

The pattern elements have the following meaning:

? Matches any single character.

* Matches any sequence of zero or more characters.

[..] Matches any of the characters inside the brackets. Ranges of characters can be specified by separating two characters by a ‘-’ (e.g. "[a0-9]" matches the letter ‘a’ or any digit). In order to represent itself, a ‘-’ must either be quoted or the first or last character in the character list. Similarly, a ‘]’ must be quoted or the first character in the list if it is to represent itself instead of the end of the list. Also, a ‘!’ appearing at the start of the list has special meaning (see below), so to represent itself it must be quoted or appear later in the list.

Within a bracket expression, the name of a *character class* enclosed in '[:' and ':]' stands for the list of all characters belonging to that class. Supported character classes:

alnum	alpha	blank	cntrl	digit	graph
lower	print	punct	space	upper	xdigit

These match characters using the macros specified in isalnum(3), isalpha(3), and so on. A character class may not be used as an endpoint of a range.

[!..] Like [..], except it matches any character not inside the brackets.

\ Matches the character following it verbatim. This is useful to quote the special characters '?', '*', '[', and '\' such that they lose their special meaning. For example, the pattern "*\[x]\?" matches the string "*[x]?".

Note that when matching a pathname, the path separator '/', is not matched by a '?', or '*', character or by a "[..]" sequence. Thus, */usr/*/bin* would match */usr/local/bin* but not */usr/bin*.

SEE ALSO

fnmatch(3), glob(3), re_format(7)

HISTORY

In early versions of UNIX, a dedicated program, */etc/glob*, was used to perform pattern expansion and pass the results to a command. In Version 7 AT&T UNIX, with the introduction of the Bourne shell, this functionality was incorporated into the shell itself.

NAME

re_format - POSIX regular expressions

DESCRIPTION

Regular expressions (REs), as defined in IEEE Std 1003.1-2004 ("POSIX.1"), come in two forms: basic regular expressions (BREs) and extended regular expressions (EREs). Both forms of regular expressions are supported by the interfaces described in regex(3). Applications dealing with regular expressions may use one or the other form (or indeed both). For example, ed(1) uses BREs, whilst egrep(1) talks EREs. Consult the manual page for the specific application to find out which it uses.

POSIX leaves some aspects of RE syntax and semantics open; '**' marks decisions on these aspects that may not be fully portable to other POSIX implementations.

This manual page first describes regular expressions in general, specifically extended regular expressions, and then discusses differences between them and basic regular expressions.

EXTENDED REGULAR EXPRESSIONS

An ERE is one** or more non-empty** *branches*, separated by 'l'. It matches anything that matches one of the branches.

A branch is one** or more *pieces*, concatenated. It matches a match for the first, followed by a match for the second, etc.

A piece is an *atom* possibly followed by a single** '*', '+', '?', or *bound*. An atom followed by '*' matches a sequence of 0 or more matches of the atom. An atom followed by '+' matches a sequence of

1 or more matches of the atom. An atom followed by '?' matches a sequence of 0 or 1 matches of the atom.

A bound is '{' followed by an unsigned decimal integer, possibly followed by ',' possibly followed by another unsigned decimal integer, always followed by '}'. The integers must lie between 0 and RE_DUP_MAX (255**) inclusive, and if there are two of them, the first may not exceed the second. An atom followed by a bound containing one integer *i* and no comma matches a sequence of exactly *i* matches of the atom. An atom followed by a bound containing one integer *i* and a comma matches a sequence of *i* or more matches of the atom. An atom followed by a bound containing two integers *i* and *j* matches a sequence of *i* through *j* (inclusive) matches of the atom.

An atom is a regular expression enclosed in '()' (matching a part of the regular expression), an empty set of '()' (matching the null string)**, a *bracket expression* (see below), '.' (matching any single character), '^' (matching the null string at the beginning of a line), '$' (matching the null string at the end of a line), a '\' followed by one of the characters '^.[$()|*+?{\' (matching that character taken as an ordinary character), a '\' followed by any other character** (matching that character taken as an ordinary character, as if the '\' had not been present**), or a single character with no other significance (matching that character). A '{' followed by a character other than a digit is an ordinary character, not the beginning of a bound**. It is illegal to end an RE with '\'.

A bracket expression is a list of characters enclosed in '[]'. It normally matches any single character from the list (but see below). If the list begins with '^', it matches any single character *not* from the rest of the list (but see below). If two characters in the list are

separated by '-', this is shorthand for the full *range* of characters between those two (inclusive) in the collating sequence, e.g. '[0-9]' in ASCII matches any decimal digit. It is illegal** for two ranges to share an endpoint, e.g. 'a-c-e'. Ranges are very collating-sequence-dependent, and portable programs should avoid relying on them.

To include a literal ']' in the list, make it the first character (following a possible '^'). To include a literal '-', make it the first or last character, or the second endpoint of a range. To use a literal '-' as the first endpoint of a range, enclose it in '[.' and '.]' to make it a collating element (see below). With the exception of these and some combinations using '[' (see next paragraphs), all other special characters, including '\', lose their special significance within a bracket expression.

Within a bracket expression, a collating element (a character, a multi-character sequence that collates as if it were a single character, or a collating-sequence name for either) enclosed in '[.' and '.]' stands for the sequence of characters of that collating element. The sequence is a single element of the bracket expression's list. A bracket expression containing a multi-character collating element can thus match more than one character, e.g. if the collating sequence includes a 'ch' collating element, then the RE '[[.ch.]]*c' matches the first five characters of 'chchcc'.

Within a bracket expression, a collating element enclosed in '[=' and '=]' is an equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself. (If there are no other equivalent collating elements, the treatment is as if the enclosing delimiters were '[.' and '.]'.) For example, if 'x' and 'y' are the members of an equivalence class, then '[[=x=]]', '[[=y=]]', and

'[xy]' are all synonymous. An equivalence class may not** be an endpoint of a range.

Within a bracket expression, the name of a *character class* enclosed in '[:' and ':]' stands for the list of all characters belonging to that class. Standard character class names are:

alnum	digit	punct
alpha	graph	space
blank	lower	upper
cntrl	print	xdigit

These stand for the character classes defined in isalnum(3), isalpha(3), and so on. A character class may not be used as an endpoint of a range.

There are two special cases** of bracket expressions: the bracket expressions '[[:<:]]' and '[[:>:]]' match the null string at the beginning and end of a word, respectively. A word is defined as a sequence of characters starting and ending with a word character which is neither preceded nor followed by word characters. A word character is an *alnum* character (as defined by isalnum(3)) or an underscore. This is an extension, compatible with but not specified by POSIX, and should be used with caution in software intended to be portable to other systems. The additional word delimiters '\<' and '\>' are provided to ease compatibility with traditional SVR4 systems but are not portable and should be avoided.

In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, it

matches the longest. Subexpressions also match the longest possible substrings, subject to the constraint that the whole match be as long as possible, with subexpressions starting earlier in the RE taking priority over ones starting later. Note that higher-level subexpressions thus take priority over their lower-level component subexpressions.

Match lengths are measured in characters, not collating elements. A null string is considered longer than no match at all. For example, ‘bb*’ matches the three middle characters of ‘abbbc’; ‘(wee|week)(knights|nights)’ matches all ten characters of ‘weeknights’; when ‘(.*).*’ is matched against ‘abc’, the parenthesized subexpression matches all three characters; and when ‘(a*)*’ is matched against ‘bc’, both the whole RE and the parenthesized subexpression match the null string.

If case-independent matching is specified, the effect is much as if all case distinctions had vanished from the alphabet. When an alphabetic that exists in multiple cases appears as an ordinary character outside a bracket expression, it is effectively transformed into a bracket expression containing both cases, e.g. ‘x’ becomes ‘[xX]’. When it appears inside a bracket expression, all case counterparts of it are added to the bracket expression, so that, for example, ‘[x]’ becomes ‘[xX]’ and ‘[^x]’ becomes ‘[^xX]’.

No particular limit is imposed on the length of REs**. Programs intended to be portable should not employ REs longer than 256 bytes, as an implementation can refuse to accept such REs and remain POSIX-compliant.

The following is a list of extended regular expressions:

c Any character *c* not listed below matches itself.

c Any backslash-escaped character *c* matches itself.

. Matches any single character that is not a newline (‘\n’).

[*char-class*]
Matches any single character in *char-class*. To include a ‘]’ in *char-class*, it must be the first character. A range of characters may be specified by separating the end characters of the range with a ‘-’; e.g. *a-z* specifies the lower case characters. The following literal expressions can also be used in *char-class* to specify sets of characters:

[:alnum:] [:cntrl:] [:lower:] [:space:]
[:alpha:] [:digit:] [:print:] [:upper:]
[:blank:] [:graph:] [:punct:] [:xdigit:]

If ‘-’ appears as the first or last character of *char-class*, then it matches itself. All other characters in *char-class* match themselves.

Patterns in *char-class* of the form [.*col-elm*.] or [=*col-elm*=], where *col-elm* is a collating element, are interpreted according to setlocale(3) (not currently supported).

[^*char-class*]
Matches any single character, other than newline, not in *char-class*. *char-class* is defined as above.

^ If ‘^’ is the first character of a regular expression, then it

anchors the regular expression to the beginning of a line. Otherwise, it matches itself.

$ If '$' is the last character of a regular expression, it anchors the regular expression to the end of a line. Otherwise, it matches itself.

[[:<:]] Anchors the single character regular expression or subexpression immediately following it to the beginning of a word.

[[:>:]] Anchors the single character regular expression or subexpression immediately preceding it to the end of a word.

(*re*) Defines a subexpression *re*. Any set of characters enclosed in parentheses matches whatever the set of characters without parentheses matches (that is a long-winded way of saying the constructs '(re)' and 're' match identically).

* Matches the single character regular expression or subexpression immediately preceding it zero or more times. If '*' is the first character of a regular expression or subexpression, then it matches itself. The '*' operator sometimes yields unexpected results. For example, the regular expression *b** matches the beginning of the string "abbb" (as opposed to the substring "bbb"), since a null match is the only leftmost match.

+ Matches the singular character regular expression or subexpression immediately preceding it one or more times.

? Matches the singular character regular expression or subexpression immediately preceding it 0 or 1 times.

{*n*,*m*} {*n*,} {*n*}
Matches the single character regular expression or subexpression immediately preceding it at least *n* and at most *m* times. If *m* is omitted, then it matches at least *n* times. If the comma is also omitted, then it matches exactly *n* times.

| Used to separate patterns. For example, the pattern 'cat|dog' matches either 'cat' or 'dog'.

BASIC REGULAR EXPRESSIONS

Basic regular expressions differ in several respects:

- '|', '+', and '?' are ordinary characters and there is no equivalent for their functionality.
- The delimiters for bounds are '\{' and '\}', with '{' and '}' by themselves ordinary characters.
- The parentheses for nested subexpressions are '\(' and '\)', with '(' and ')' by themselves ordinary characters.
- '^' is an ordinary character except at the beginning of the RE or** the beginning of a parenthesized subexpression.
- '$' is an ordinary character except at the end of the RE or** the end of a parenthesized subexpression.
- '*' is an ordinary character if it appears at the beginning of the

RE or the beginning of a parenthesized subexpression (after a possible leading '^').

- Finally, there is one new type of atom, a *back-reference*: '\' followed by a non-zero decimal digit *d* matches the same sequence of characters matched by the *d*th parenthesized subexpression (numbering subexpressions by the positions of their opening parentheses, left to right), so that, for example, '\([bc]\)\1' matches 'bb' or 'cc' but not 'bc'.

The following is a list of basic regular expressions:

c Any character *c* not listed below matches itself.

c Any backslash-escaped character *c*, except for '{', '}', '(', and ')', matches itself.

. Matches any single character that is not a newline ('\n').

[*char-class*]

Matches any single character in *char-class*. To include a ']' in *char-class*, it must be the first character. A range of characters may be specified by separating the end characters of the range with a '-'; e.g. *a-z* specifies the lower case characters. The following literal expressions can also be used in *char-class* to specify sets of characters:

[:alnum:] [:cntrl:] [:lower:] [:space:]
[:alpha:] [:digit:] [:print:] [:upper:]
[:blank:] [:graph:] [:punct:] [:xdigit:]

If '-' appears as the first or last character of *char-class*, then it matches itself. All other characters in *char-class* match themselves.

Patterns in *char-class* of the form [.*col-elm*.] or [=*col-elm*=], where *col-elm* is a collating element, are interpreted according to setlocale(3) (not currently supported).

[^*char-class*]
Matches any single character, other than newline, not in *char-class*. *char-class* is defined as above.

^ If '^' is the first character of a regular expression, then it anchors the regular expression to the beginning of a line. Otherwise, it matches itself.

$ If '$' is the last character of a regular expression, it anchors the regular expression to the end of a line. Otherwise, it matches itself.

[[:<:]] Anchors the single character regular expression or subexpression immediately following it to the beginning of a word.

[[:>:]] Anchors the single character regular expression or subexpression immediately following it to the end of a word.

\(*re*\) Defines a subexpression *re*. Subexpressions may be nested. A subsequent backreference of the form *n*, where *n* is a number in the range [1,9], expands to the text matched by the *n*th subexpression. For example, the regular expression

\(.*\)*1* matches any string consisting of identical adjacent substrings. Subexpressions are ordered relative to their left delimiter.

* Matches the single character regular expression or subexpression immediately preceding it zero or more times. If '*' is the first character of a regular expression or subexpression, then it matches itself. The '*' operator sometimes yields unexpected results. For example, the regular expression *b** matches the beginning of the string "abbb" (as opposed to the substring "bbb"), since a null match is the only leftmost match.

\{*n*,*m*\} \{*n*,\} \{*n*\}
Matches the single character regular expression or subexpression immediately preceding it at least *n* and at most *m* times. If *m* is omitted, then it matches at least *n* times. If the comma is also omitted, then it matches exactly *n* times.

SEE ALSO

regex(3)