NAME

bash, :, ., alias, bg, bind, break, builtin, bye, case, cd, command, continue, declare, dirs, echo, enable, eval, exec, exit, export, fc, fg, for, getopts, hash, help, history, if, jobs, kill, let, local, logout, popd, pushd, pwd, read, readonly, return, set, shift, source, suspend, test, times, trap, type, typeset, ulimit, umask, unalias, unset, until, wait, while − bash built-in commands, see **bash**(1)

BASH BUILTIN COMMANDS

: [*arguments*]

No effect; the command does nothing beyond expanding *arguments* and performing any specified redirections. A zero exit code is returned.

. *filename* [*arguments*]
source *filename* [*arguments*]

Read and execute commands from *filename* in the current shell environment and return the exit status of the last command executed from *filename*. If *filename* does not contain a slash, pathnames in **PATH** are used to find the directory containing *filename*. The file searched for in **PATH** need not be executable. The current directory is searched if no file is found in **PATH**. If any *arguments* are supplied, they become the positional parameters when *file* is executed. Otherwise the positional parameters are unchanged. The return status is the status of the last command exited within the script (0 if no commands are executed), and false if *filename* is not found.

alias [*name*[=*value*] ...]

Alias with no arguments prints the list of aliases in the form *name*=*value* on standard output. When arguments are supplied, an alias is defined for each *name* whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution when the alias is expanded. For each *name* in the argument list for which no *value* is supplied, the name and value of the alias is printed. **Alias** returns true unless a *name* is given for which no alias has been defined.

bg [*jobspec*]

Place *jobspec* in the background, as if it had been started with **&**. If *jobspec* is not present, the shell's notion of the *current job* is used. **bg** *jobspec* returns 0 unless run when job control is disabled or, when run with job control enabled, if *jobspec* was not found or started without job control.

bind [**−m** *keymap*] [**−lvd**] [**-q** *name*]
bind [**−m** *keymap*] **-f** *filename*
bind [**−m** *keymap*] *keyseq*:*function-name*

Display current **readline** key and function bindings, or bind a key sequence to a **readline** function or macro. The binding syntax accepted is identical to that of *.inputrc*, but each binding must be passed as a separate argument; e.g., '"\C-x\C-r": re−read−init−file'. Options, if supplied, have the following meanings:

−m *keymap*

Use *keymap* as the keymap to be affected by the subsequent bindings. Acceptable *keymap* names are *emacs, emacs-standard, emacs-meta, emacs-ctlx, vi, vi-move, vi-command*, and *vi-insert*. *vi* is equivalent to *vi-command*; *emacs* is equivalent to *emacs-standard*.

−l List the names of all **readline** functions

−v List current function names and bindings

−d Dump function names and bindings in such a way that they can be re-read

−f *filename*

Read key bindings from *filename*

−q *function*

Query about which keys invoke the named *function*

The return value is 0 unless an unrecognized option is given or an error occurred.

break [*n*]

Exit from within a **for**, **while**, or **until** loop. If *n* is specified, break *n* levels. *n* must be ≥ 1. If *n* is greater than the number of enclosing loops, all enclosing loops are exited. The return value is 0 unless the shell is not executing a loop when **break** is executed.

builtin *shell−builtin* [*arguments*]

Execute the specified shell builtin, passing it *arguments*, and return its exit status. This is useful when you wish to define a function whose name is the same as a shell builtin, but need the functionality of the builtin within the function itself. The **cd** builtin is commonly redefined this way. The return status is false if *shell−builtin* is not a shell builtin command.

cd [*dir*] Change the current directory to *dir*. The variable **HOME** is the default *dir*. The variable **CDPATH** defines the search path for the directory containing *dir*. Alternative directory names are separated by a colon (:). A null directory name in **CDPATH** is the same as the current directory, i.e., "**.**". If *dir* begins with a slash (*/*), then **CDPATH** is not used. An argument of **−** is equivalent to **$OLDPWD**. The return value is true if the directory was successfully changed; false otherwise.

command [**-pVv**] *command* [*arg ...*]

Run *command* with *args* suppressing the normal shell function lookup. Only builtin commands or commands found in the **PATH** are executed. If the **−p** option is given, the search for *command* is performed using a default value for **PATH** that is guaranteed to find all of the standard utilities. If either the **−V** or **−v** option is supplied, a description of *command* is printed. The **−v** option causes a single word indicating the command or pathname used to invoke *command* to be printed; the **−V** option produces a more verbose description. An argument of **−−** disables option checking for the rest of the arguments. If the **−V** or **−v** option is supplied, the exit status is 0 if *command* was found, and 1 if not. If neither option is supplied and an error occurred or *command* cannot be found, the exit status is 127. Otherwise, the exit status of the **command** builtin is the exit status of *command*.

continue [*n*]

Resume the next iteration of the enclosing **for**, **while**, or **until** loop. If *n* is specified, resume at the *n*th enclosing loop. *n* must be ≥ 1. If *n* is greater than the number of enclosing loops, the last enclosing loop (the 'top−level' loop) is resumed. The return value is 0 unless the shell is not executing a loop when **continue** is executed.

declare [**−frxi**] [*name*[=*value*]]
typeset [**−frxi**] [*name*[=*value*]]

Declare variables and/or give them attributes. If no *name*s are given, then display the values of variables instead. The options can be used to restrict output to variables with the specified attribute.

- **−f** Use function names only
- **−r** Make *name*s readonly. These names cannot then be assigned values by subsequent assignment statements.
- **−x** Mark *name*s for export to subsequent commands via the environment.
- **−i** The variable is treated as an integer; arithmetic evaluation (see **ARITHMETIC EVALUATION**) is performed when the variable is assigned a value.

Using '+' instead of '−' turns off the attribute instead. When used in a function, makes *name*s local, as with the **local** command. The return value is 0 unless an illegal option is encountered, an attempt is made to define a function using "-f foo=bar", one of the *names* is not a legal shell variable name, an attempt is made to turn off readonly status for a readonly variable, or an attempt is made to display a non-existant function with -f.

dirs [-l] [+/−n]

Display the list of currently remembered directories. Directories are added to the list with the **pushd** command; the **popd** command moves back up through the list.

- **+n** displays the *n*th entry counting from the left of the list shown by **dirs** when invoked without options, starting with zero.

−n displays the *n*th entry counting from the right of the list shown by **dirs** when invoked without options, starting with zero.

−l produces a longer listing; the default listing format uses a tilde to denote the home directory.

The return value is 0 unless an illegal option is supplied or *n* indexes beyond the end of the directory stack.

echo [**−neE**] [*arg* ...]

Output the *arg*s, separated by spaces. The return status is always 0. If **−n** is specified, the trailing newline is suppressed. If the **−e** option is given, interpretation of the following backslash-escaped characters is enabled. The **−E** option disables the interpretation of these escape characters, even on systems where they are interpreted by default.

\a alert (bell)
\b backspace
\c suppress trailing newline
\f form feed
\n new line
\r carriage return
\t horizontal tab
\v vertical tab
**** backslash
\nnn the character whose ASCII code is *nnn* (octal)

enable [**−n**] [**−all**] [*name* ...]

Enable and disable builtin shell commands. This allows the execution of a disk command which has the same name as a shell builtin without specifying a full pathname. If **−n** is used, each *name* is disabled; otherwise, *names* are enabled. For example, to use the **test** binary found via the **PATH** instead of the shell builtin version, type ''enable -n test''. If no arguments are given, a list of all enabled shell builtins is printed. If only **−n** is supplied, a list of all disabled builtins is printed. If only **−all** is supplied, the list printed includes all builtins, with an indication of whether or not each is enabled. **enable** accepts **−a** as a synonym for **−all**. The return value is 0 unless a *name* is not a shell builtin.

eval [*arg* ...]

The *arg*s are read and concatenated together into a single command. This command is then read and executed by the shell, and its exit status is returned as the value of the **eval** command. If there are no *args*, or only null arguments, **eval** returns true.

exec [[**−**] *command* [*arguments*]]

If *command* is specified, it replaces the shell. No new process is created. The *arguments* become the arguments to *command*. If the first argument is **−**, the shell places a dash in the zeroth arg passed to *command*. This is what login does. If the file cannot be executed for some reason, a non-interactive shell exits, unless the shell variable **no_exit_on_failed_exec** exists, in which case it returns failure. An interactive shell returns failure if the file cannot be executed. If *command* is not specified, any redirections take effect in the current shell, and the return status is 0.

exit [*n*] Cause the shell to exit with a status of *n*. If *n* is omitted, the exit status is that of the last command executed. A trap on **EXIT** is executed before the shell terminates.

export [**−nf**] [*name*[=*word*]] ...
export −p

The supplied *names* are marked for automatic export to the environment of subsequently executed commands. If the **−f** option is given, the *names* refer to functions. If no *names* are given, or if the **−p** option is supplied, a list of all names that are exported in this shell is printed. The **−n** option causes the export property to be removed from the named variables. An argument of **−−** disables option checking for the rest of the arguments. **export** returns an exit status of 0 unless an illegal option is encountered, one of the *names* is not a legal shell variable name, or **−f** is supplied with a *name* that is not a function.

fc [**−e** *ename*] [**−nlr**] [*first*] [*last*]
fc −s [*pat*=*rep*] [*cmd*]

Fix Command. In the first form, a range of commands from *first* to *last* is selected from the history list. *First* and *last* may be specified as a string (to locate the last command beginning with that string) or as a number (an index into the history list, where a negative number is used as an offset from the current command number). If *last* is not specified it is set to the current command for listing (so that **fc −l −10** prints the last 10 commands) and to *first* otherwise. If *first* is not specified it is set to the previous command for editing and −16 for listing.

The **−n** flag suppresses the command numbers when listing. The **−r** flag reverses the order of the commands. If the **−l** flag is given, the commands are listed on standard output. Otherwise, the editor given by *ename* is invoked on a file containing those commands. If *ename* is not given, the value of the **FCEDIT** variable is used, and the value of **EDITOR** if **FCEDIT** is not set. If neither variable is set, is used. When editing is complete, the edited commands are echoed and executed.

In the second form, *command* is re-executed after each instance of *pat* is replaced by *rep*. A useful alias to use with this is ‘‘r=fc −s’’, so that typing ‘‘r cc’’ runs the last command beginning with ‘‘cc’’ and typing ‘‘r’’ re-executes the last command.

If the first form is used, the return value is 0 unless an illegal option is encountered or *first* or *last* specify history lines out of range. If the **−e** option is supplied, the return value is the value of the last command executed or failure if an error occurs with the temporary file of commands. If the second form is used, the return status is that of the command re-executed, unless *cmd* does not specify a valid history line, in which case **fc** returns failure.

fg [*jobspec*]

Place *jobspec* in the foreground, and make it the current job. If *jobspec* is not present, the shell's notion of the *current job* is used. The return value is that of the command placed into the foreground, or failure if run when job control is disabled or, when run with job control enabled, if *jobspec* does not specify a valid job or *jobspec* specifies a job that was started without job control.

getopts *optstring name* [*args*]

getopts is used by shell procedures to parse positional parameters. *optstring* contains the option letters to be recognized; if a letter is followed by a colon, the option is expected to have an argument, which should be separated from it by white space. Each time it is invoked, **getopts** places the next option in the shell variable *name*, initializing *name* if it does not exist, and the index of the next argument to be processed into the variable **OPTIND**. **OPTIND** is initialized to 1 each time the shell or a shell script is invoked. When an option requires an argument, **getopts** places that argument into the variable **OPTARG**. The shell does not reset **OPTIND** automatically; it must be manually reset between multiple calls to **getopts** within the same shell invocation if a new set of parameters is to be used.

getopts can report errors in two ways. If the first character of *optstring* is a colon, *silent* error reporting is used. In normal operation diagnostic messages are printed when illegal options or missing option arguments are encountered. If the variable **OPTERR** is set to 0, no error message will be displayed, even if the first character of *optstring* is not a colon.

If an illegal option is seen, **getopts** places ? into *name* and, if not silent, prints an error message and unsets **OPTARG**. If **getopts** is silent, the option character found is placed in **OPTARG** and no diagnostic message is printed.

If a required argument is not found, and **getopts** is not silent, a question mark (**?**) is placed in *name*, **OPTARG** is unset, and a diagnostic message is printed. If **getopts** is silent, then a colon (**:**) is placed in *name* and **OPTARG** is set to the option character found.

getopts normally parses the positional parameters, but if more arguments are given in *args*,

getopts parses those instead. **getopts** returns true if an option, specified or unspecified, is found. It returns false if the end of options is encountered or an error occurs.

hash [**−r**] [*name*]
For each *name*, the full pathname of the command is determined and remembered. The **−r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is printed. An argument of **−−** disables option checking for the rest of the arguments. The return status is true unless a *name* is not found or an illegal option is supplied.

help [*pattern*]
Display helpful information about builtin commands. If *pattern* is specified, **help** gives detailed help on all commands matching *pattern*; otherwise a list of the builtins is printed. The return status is 0 unless no command matches *pattern*.

history [*n*]
history −rwan [*filename*]
With no options, display the command history list with line numbers. Lines listed with a ***** have been modified. An argument of *n* lists only the last *n* lines. If a non-option argument is supplied, it is used as the name of the history file; if not, the value of **HISTFILE** is used. Options, if supplied, have the following meanings:

−a Append the ''new'' history lines (history lines entered since the beginning of the current **bash** session) to the history file

−n Read the history lines not already read from the history file into the current history list. These are lines appended to the history file since the beginning of the current **bash** session.

−r Read the contents of the history file and use them as the current history

−w Write the current history to the history file, overwriting the history file's contents.

The return value is 0 unless an illegal option is encountered or an error occurs while reading or writing the history file.

jobs [**−lnp**] [*jobspec ...*]
jobs −x *command* [*args ...*]
The first form lists the active jobs. The **−l** option lists process IDs in addition to the normal information; the **−p** option lists only the process ID of the job's process group leader. The **−n** option displays only jobs that have changed status since last notified. If *jobspec* is given, output is restricted to information about that job. The return status is 0 unless an illegal option is encountered or an illegal *jobspec* is supplied.

If the **−x** option is supplied, **jobs** replaces any *jobspec* found in *command* or *args* with the corresponding process group ID, and executes *command* passing it *args*, returning its exit status.

kill [**-s sigspec | −sigspec**] [*pid | jobspec*] ...
kill −l [*signum*]
Send the signal named by *sigspec* to the processes named by *pid* or *jobspec*. *sigspec* is either a signal name such as **SIGKILL** or a signal number. If *sigspec* is a signal name, the name is case insensitive and may be given with or without the **SIG** prefix. If *sigspec* is not present, then **SIGTERM** is assumed. An argument of **−l** lists the signal names. If any arguments are supplied when **−l** is given, the names of the specified signals are listed, and the return status is 0. An argument of **−−** disables option checking for the rest of the arguments. **kill** returns true if at least one signal was successfully sent, or false if an error occurs or an illegal option is encountered.

let *arg* [*arg ...*]
Each *arg* is an arithmetic expression to be evaluated (see **ARITHMETIC EVALUATION**). If the last *arg* evaluates to 0, **let** returns 1; 0 is returned otherwise.

local [*name*[=*value*] ...]
For each argument, create a local variable named *name*, and assign it *value*. When **local** is used within a function, it causes the variable *name* to have a visible scope restricted to that function and

its children. With no operands, **local** writes a list of local variables to the standard output. It is an error to use **local** when not within a function. The return status is 0 unless **local** is used outside a function, or an illegal *name* is supplied.

logout Exit a login shell.

popd [**+/−n**]

Removes entries from the directory stack. With no arguments, removes the top directory from the stack, and performs a **cd** to the new top directory.

+n removes the *n*th entry counting from the left of the list shown by **dirs**, starting with zero. For example: ''popd +0'' removes the first directory, ''popd +1'' the second.

−n removes the *n*th entry counting from the right of the list shown by **dirs**, starting with zero. For example: ''popd -0'' removes the last directory, ''popd -1'' the next to last.

If the **popd** command is successful, a **dirs** is performed as well, and the return status is 0. **popd** returns false if an illegal option is encountered, the directory stack is empty, a non-existent directory stack entry is specified, or the directory change fails.

pushd [*dir*]
pushd +/−n

Adds a directory to the top of the directory stack, or rotates the stack, making the new top of the stack the current working directory. With no arguments, exchanges the top two directories and returns 0, unless the directory stack is empty.

+n Rotates the stack so that the *n*th directory (counting from the left of the list shown by **dirs**) is at the top.

−n Rotates the stack so that the *n*th directory (counting from the right) is at the top.

dir adds *dir* to the directory stack at the top, making it the new current working directory.

If the **pushd** command is successful, a **dirs** is performed as well. If the first form is used, **pushd** returns 0 unless the cd to *dir* fails. With the second form, **pushd** returns 0 unless the directory stack is empty, a non-existant directory stack element is specified, or the directory change to the specified new current directory fails.

pwd Print the absolute pathname of the current working directory. The path printed contains no symbolic links if the **−P** option to the **set** builtin command is set. See also the description of **nolinks** under **Shell Variables** above). The return status is 0 unless an error occurs while reading the pathname of the current directory.

read [**−r**] [*name ...*]

One line is read from the standard input, and the first word is assigned to the first *name*, the second word to the second *name*, and so on, with leftover words assigned to the last *name*. Only the characters in **IFS** are recognized as word delimiters. If no *names* are supplied, the line read is assigned to the variable **REPLY**. The return code is zero, unless end-of-file is encountered. If the **−r** option is given, a backslash-newline pair is not ignored, and the backslash is considered to be part of the line.

readonly [**−f**] [*name ...*]
readonly -p

The given *names* are marked readonly and the values of these *names* may not be changed by subsequent assignment. If the **−f** option is supplied, the functions corresponding to the *names* are so marked. If no arguments are given, or if the **−p** option is supplied, a list of all readonly names is printed. An argument of **−−** disables option checking for the rest of the arguments. The return status is 0 unless an illegal option is encountered, one of the *names* is not a legal shell variable name, or **−f** is supplied with a *name* that is not a function.

return [*n*]

Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed in the function body. If used outside a function, but during execution of a script by the **.** (**source**) command, it causes the shell to stop executing that script and return either *n* or the exit status of the last command executed within the script as the exit

status of the script. If used outside a function and not during execution of a script by **.**, the return status is false.

set [**−−abefhkmnptuvxldCHP**] [**-o** *option*] [*arg ...*]

−a Automatically mark variables which are modified or created for export to the environment of subsequent commands.

−b Cause the status of terminated background jobs to be reported immediately, rather than before the next primary prompt. (Also see **notify** under **Shell Variables** above).

−e Exit immediately if a *simple-command* (see **SHELL GRAMMAR** above) exits with a non−zero status. The shell does not exit if the command that fails is part of an *until* or *while* loop, part of an *if* statement, part of a **&&** or | | list, or if the command's return value is being inverted via **!**.

−f Disable pathname expansion.

−h Locate and remember function commands as functions are defined. Function commands are normally looked up when the function is executed.

−k All keyword arguments are placed in the environment for a command, not just those that precede the command name.

−m Monitor mode. Job control is enabled. This flag is on by default for interactive shells on systems that support it (see **JOB CONTROL** above). Background processes run in a separate process group and a line containing their exit status is printed upon their completion.

−n Read commands but do not execute them. This may be used to check a shell script for syntax errors. This is ignored for interactive shells.

−o *option-name*

The *option-name* can be one of the following:

allexport
Same as **−a**.

braceexpand
The shell performs brace expansion (see **Brace Expansion** above). This is on by default.

emacs Use an emacs-style command line editing interface. This is enabled by default when the shell is interactive, unless the shell is started with the **−nolineediting** option.

errexit Same as **−e**.

histexpand
Same as **−H**.

ignoreeof
The effect is as if the shell command 'IGNOREEOF=10' had been executed (see **Shell Variables** above).

interactive−comments
Allow a word beginning with **#** to cause that word and all remaining characters on that line to be ignored in an interactive shell (see **COMMENTS** above).

monitor Same as **−m**.

noclobber
Same as **−C**.

noexec Same as **−n**.

noglob Same as **−f**.

nohash Same as **−d**.

notify Same as **−b**.

nounset Same as **−u**.

physical Same as **−P**.

posix Change the behavior of bash where the default operation differs from the Posix 1003.2 standard to match the standard.

privileged
Same as **−p**.
verbose Same as **−v**.
vi Use a vi-style command line editing interface.
xtrace Same as **−x**.
If no *option-name* is supplied, the values of the current options are printed.

−p Turn on *privileged* mode. In this mode, the **$ENV** file is not processed, and shell functions are not inherited from the environment. This is enabled automatically on startup if the effective user (group) id is not equal to the real user (group) id. Turning this option off causes the effective user and group ids to be set to the real user and group ids.

−t Exit after reading and executing one command.

−u Treat unset variables as an error when performing parameter expansion. If expansion is attempted on an unset variable, the shell prints an error message, and, if not interactive, exits with a non−zero status.

−v Print shell input lines as they are read.

−x After expanding each *simple-command*, **bash** displays the expanded value of **PS4**, followed by the command and its expanded arguments.

−l Save and restore the binding of *name* in a **for** *name* [in **word**] command (see **SHELL GRAMMAR** above).

−d Disable the hashing of commands that are looked up for execution. Normally, commands are remembered in a hash table, and once found, do not have to be looked up again.

−C The effect is as if the shell command 'noclobber=' had been executed (see **Shell Variables** above).

−H Enable **!** style history substitution. This flag is on by default when the shell is interactive.

−P If set, do not follow symbolic links when performing commands such as **cd** which change the current directory. The physical directory is used instead.

−− If no arguments follow this flag, then the positional parameters are unset. Otherwise, the positional parameters are set to the *arg*s, even if some of them begin with a **−**.

− Signal the end of options, cause all remaining *arg*s to be assigned to the positional parameters. The **−x** and **−v** options are turned off. If there are no *arg*s, the positional parameters remain unchanged.

The flags are off by default unless otherwise noted. Using + rather than − causes these flags to be turned off. The flags can also be specified as options to an invocation of the shell. The current set of flags may be found in **$−**. After the option arguments are processed, the remaining *n arg*s are treated as values for the positional parameters and are assigned, in order, to **$1**, **$2**, **...** **$***n*. If no options or *arg*s are supplied, all shell variables are printed. The return status is always true unless an illegal option is encountered.

shift [*n*]
The positional parameters from *n*+1 ... are renamed to **$1 ....** If *n* is not given, it is assumed to be 1. The exit status is 1 if *n* is greater than **$#**; otherwise 0.

suspend [**−f**]
Suspend the execution of this shell until it receives a **SIGCONT** signal. The **−f** option says not to complain if this is a login shell; just suspend anyway. The return status is 0 unless the shell is a login shell and **−f** is not supplied, or if job control is not enabled.

test *expr*
[*expr* **]** Return a status of 0 (true) or 1 (false) depending on the evaluation of the conditional expression *expr*. Expressions may be unary or binary. Unary expressions are often used to examine the status of a file. There are string operators and numeric comparison operators as well. Each operator and operand must be a separate argument. If *file* is of the form /dev/fd/*n*, then file descriptor *n* is checked.

−b *file* True if *file* exists and is block special.
−c *file* True if *file* exists and is character special.
−d *file* True if *file* exists and is a directory.
−e *file* True if *file* exists.
−f *file* True if *file* exists and is a regular file.
−g *file* True if *file* exists and is set-group-id.
−k *file* True if *file* has its ‘‘sticky’’ bit set.
−L *file* True if *file* exists and is a symbolic link.
−p *file* True if *file* exists and is a named pipe.
−r *file* True if *file* exists and is readable.
−s *file* True if *file* exists and has a size greater than zero.
−S *file* True if *file* exists and is a socket.
−t *fd* True if *fd* is opened on a terminal.
−u *file* True if *file* exists and its set-user-id bit is set.
−w *file* True if *file* exists and is writable.
−x *file* True if *file* exists and is executable.
−O *file* True if *file* exists and is owned by the effective user id.
−G *file* True if *file* exists and is owned by the effective group id.
file1 **−nt** *file2*
True if *file1* is newer (according to modification date) than *file2*.
file1 **−ot** *file2*
True if *file1* is older than file2.
file1 **−ef** *file*
True if *file1* and *file2* have the same device and inode numbers.
−z *string*
True if the length of *string* is zero.
−n *string*
string True if the length of *string* is non−zero.
string1 **=** *string2*
True if the strings are equal.
string1 **!=** *string2*
True if the strings are not equal.
! *expr* True if *expr* is false.
expr1 **−a** *expr2*
True if both *expr1* AND *expr2* are true.
expr1 **−o** *expr2*
True if either *expr1* OR *expr2* is true.
arg1 **OP** *arg2*
OP is one of **−eq**, **−ne**, **−lt**, **−le**, **−gt**, or **−ge**. These arithmetic binary operators return true if *arg1* is equal, not-equal, less-than, less-than-or-equal, greater-than, or greater-than-or-equal than *arg2*, respectively. *Arg1* and *arg2* may be positive integers, negative integers, or the special expression **−l** *string*, which evaluates to the length of *string*.

times Print the accumulated user and system times for the shell and for processes run from the shell. The return status is 0.

trap [**−l**] [*arg*] [*sigspec*]
The command *arg* is to be read and executed when the shell receives signal(s) *sigspec*. If *arg* is absent or **−**, all specified signals are reset to their original values (the values they had upon entrance to the shell). If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. *sigspec* is either a signal name defined in <*signal.h*>, or a signal number. If *sigspec* is **EXIT** (0) the command *arg* is executed on exit from the shell. With no arguments, **trap** prints the list of commands associated with each signal number. The **−l** option causes the shell to print a list of signal names and their corresponding numbers. An argument of **−−** disables option checking for the rest of the arguments. Signals ignored upon entry to the shell cannot be trapped or reset. Trapped signals are reset to their original values in a child process when it is created.

The return status is false if either the trap name or number is invalid; otherwise **trap** returns true.

type [**−all**] [**−type** | **−path**] *name* [*name ...*]
With no options, indicate how each *name* would be interpreted if used as a command name. If the **−type** flag is used, **type** prints a phrase which is one of *alias*, *keyword*, *function*, *builtin*, or *file* if *name* is an alias, shell reserved word, function, builtin, or disk file, respectively. If the name is not found, then nothing is printed, and an exit status of false is returned. If the **−path** flag is used, **type** either returns the name of the disk file that would be executed if *name* were specified as a command name, or nothing if **−type** would not return *file*. If a command is hashed, **−path** prints the hashed value, not necessarily the file that appears first in **PATH**. If the **−all** flag is used, **type** prints all of the places that contain an executable named *name*. This includes aliases and functions, if and only if the **−path** flag is not also used. The table of hashed commands is not consulted when using **−all**. **type** accepts **−a**, **−t**, and **−p** in place of **−all**, **−type**, and **−path**, respectively. An argument of **−−** disables option checking for the rest of the arguments. **type** returns true if any of the arguments are found, false if none are found.

ulimit [**−SHacdfmstpnuv** [*limit*]]
Ulimit provides control over the resources available to the shell and to processes started by it, on systems that allow such control. The value of *limit* can be a number in the unit specified for the resource, or the value **unlimited**. The **H** and **S** options specify that the hard or soft limit is set for the given resource. A hard limit cannot be increased once it is set; a soft limit may be increased up to the value of the hard limit. If neither **H** nor **S** is specified, the command applies to the soft limit. If *limit* is omitted, the current value of the soft limit of the resource is printed, unless the **H** option is given. When more than one resource is specified, the limit name and unit is printed before the value. Other options are interpreted as follows:

−a all current limits are reported
−c the maximum size of core files created
−d the maximum size of a process's data segment
−f the maximum size of files created by the shell
−m the maximum resident set size
−s the maximum stack size
−t the maximum amount of cpu time in seconds
−p the pipe size in 512-byte blocks (this may not be set)
−n the maximum number of open file descriptors (most systems do not allow this value to be set, only displayed)
−u the maximum number of processes available to a single user
−v The maximum amount of virtual memory available to the shell

An argument of **−−** disables option checking for the rest of the arguments. If *limit* is given, it is the new value of the specified resource (the **−a** option is display only). If no option is given, then **−f** is assumed. Values are in 1024-byte increments, except for **−t**, which is in seconds, **−p**, which is in units of 512-byte blocks, and **−n** and **−u**, which are unscaled values. The return status is 0 unless an illegal option is encountered, a non-numeric argument other than **unlimited** is supplied as *limit*, or an error occurs while setting a new limit.

umask [**−S**] [*mode*]
The user file-creation mask is set to *mode*. If *mode* begins with a digit, it is interpreted as an octal number; otherwise it is interpreted as a symbolic mode mask similar to that accepted by *chmod*(1). If *mode* is omitted, or if the **−S** option is supplied, the current value of the mask is printed. The **−S** option causes the mask to be printed in symbolic form; the default output is an octal number. An argument of **−−** disables option checking for the rest of the arguments. The return status is 0 if the mode was successfully changed or if no *mode* argument was supplied, and false otherwise.

unalias [**−a**] [*name ...*]
Remove *name*s from the list of defined aliases. If **−a** is supplied, all alias definitions are removed. The return value is true unless a supplied *name* is not a defined alias.

unset [**−fv**] [*name* ...]

For each *name*, remove the corresponding variable or, given the **−f** option, function. An argument of **−−** disables option checking for the rest of the arguments. Note that **PATH**, **IFS**, **PPID**, **PS1**, **PS2**, **UID**, and **EUID** cannot be unset. If any of **RANDOM**, **SECONDS**, **LINENO**, or **HISTCMD** are unset, they lose their special properties, even if they are subsequently reset. The exit status is true unless a *name* does not exist or is non-unsettable.

wait [*n*]

Wait for the specified process and return its termination status. *n* may be a process ID or a job specification; if a job spec is given, all processes in that job's pipeline are waited for. If *n* is not given, all currently active child processes are waited for, and the return status is zero. If *n* specifies a non-existant process or job, the return status is 127. Otherwise, the return status is the exit status of the last process or job waited for.

SEE ALSO

bash(1), sh(1)