NAME

bash, :, ., [, alias, bg, bind, break, builtin, caller, cd, command, compgen, complete, compopt, continue, declare, dirs, disown, echo, enable, eval, exec, exit, export, false, fc, fg, getopts, hash, help, history, jobs, kill, let, local, logout, mapfile, popd, printf, pushd, pwd, read, readonly, return, set, shift, shopt, source, suspend, test, times, trap, true, type, typeset, ulimit, umask, unalias, unset, wait − bash built-in commands, see **bash**(1)

BASH BUILTIN COMMANDS

Unless otherwise noted, each builtin command documented in this section as accepting options preceded by **−** accepts **−−** to signify the end of the options. The **:**, **true**, **false**, and **test** builtins do not accept options and do not treat **−−** specially. The **exit**, **logout**, **break**, **continue**, **let**, and **shift** builtins accept and process arguments beginning with **−** without requiring **−−**. Other builtins that accept arguments but are not specified as accepting options interpret arguments beginning with **−** as invalid options and require **−−** to prevent this interpretation.

: [*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, filenames in **PATH** are used to find the directory containing *filename*. The file searched for in **PATH** need not be executable. When **bash** is not in *posix mode*, the current directory is searched if no file is found in **PATH**. If the **sourcepath** option to the **shopt** builtin command is turned off, the **PATH** is not searched. If any *arguments* are supplied, they become the positional parameters when *filename* 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 or cannot be read.

alias [**−p**] [*name*[=*value*] ...]

Alias with no arguments or with the **−p** option prints the list of aliases in the form **alias** *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* ...]

Resume each suspended job *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, any specified *jobspec* was not found or was started without job control.

bind [**−m** *keymap*] [**−lpsvPSVX**]
bind [**−m** *keymap*] [**−q** *function*] [**−u** *function*] [**−r** *keyseq*]
bind [**−m** *keymap*] **−f** *filename*
bind [**−m** *keymap*] **−x** *keyseq*:*shell−command*
bind [**−m** *keymap*] *keyseq*:*function−name*
bind *readline−command*

Display current **readline** key and function bindings, bind a key sequence to a **readline** function or macro, or set a **readline** variable. Each non-option argument is a command as it would appear in *.inputrc*, but each binding or command 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.

−p Display **readline** function names and bindings in such a way that they can be re-read.

−P List current **readline** function names and bindings.

−s Display **readline** key sequences bound to macros and the strings they output in such a way that they can be re-read.

−S Display **readline** key sequences bound to macros and the strings they output.

−v Display **readline** variable names and values in such a way that they can be re-read.

−V List current **readline** variable names and values.

−f *filename*
Read key bindings from *filename*.

−q *function*
Query about which keys invoke the named *function*.

−u *function*
Unbind all keys bound to the named *function*.

−r *keyseq*
Remove any current binding for *keyseq*.

−x *keyseq***:***shell−command*
Cause *shell−command* to be executed whenever *keyseq* is entered. When *shell−command* is executed, the shell sets the **READLINE_LINE** variable to the contents of the **readline** line buffer and the **READLINE_POINT** variable to the current location of the insertion point. If the executed command changes the value of **READLINE_LINE** or **READLINE_POINT**, those new values will be reflected in the editing state.

−X List all key sequences bound to shell commands and the associated commands in a format that can be reused as input.

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

break [*n*]
Exit from within a **for**, **while**, **until**, or **select** 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 *n* is not greater than or equal to 1.

builtin *shell−builtin* [*arguments*]
Execute the specified shell builtin, passing it *arguments*, and return its exit status. This is useful when defining a function whose name is the same as a shell builtin, retaining the functionality of the builtin within the function. The **cd** builtin is commonly redefined this way. The return status is false if *shell−builtin* is not a shell builtin command.

caller [*expr*]
Returns the context of any active subroutine call (a shell function or a script executed with the **.** or **source** builtins). Without *expr*, **caller** displays the line number and source filename of the current subroutine call. If a non-negative integer is supplied as *expr*, **caller** displays the line number, subroutine name, and source file corresponding to that position in the current execution call stack. This extra information may be used, for example, to print a stack trace. The current frame is frame 0. The return value is 0 unless the shell is not executing a subroutine call or *expr* does not correspond to a valid position in the call stack.

cd [**−L**|[**−P** [**−e**]]] [*dir*]
Change the current directory to *dir*. if *dir* is not supplied, the value of the **HOME** shell variable is the default. Any additional arguments following *dir* are ignored. The variable **CDPATH** defines the search path for the directory containing *dir*: each directory name in **CDPATH** is searched for *dir*. Alternative directory names in **CDPATH** 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. The **−P** option causes **cd** to use the physical directory structure by resolving symbolic links while traversing *dir* and before processing instances of .. in *dir* (see also the **−P**

option to the **set** builtin command); the **−L** option forces symbolic links to be followed by resolving the link after processing instances of .. in *dir*. If .. appears in *dir*, it is processed by removing the immediately previous pathname component from *dir*, back to a slash or the beginning of *dir*. If the **−e** option is supplied with **−P**, and the current working directory cannot be successfully determined after a successful directory change, **cd** will return an unsuccessful status. An argument of **−** is converted to **$OLDPWD** before the directory change is attempted. If a non-empty directory name from **CDPATH** is used, or if **−** is the first argument, and the directory change is successful, the absolute pathname of the new working directory is written to the standard output. 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 filename used to invoke *command* to be displayed; the **−V** option produces a more verbose description. 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*.

compgen [*option*] [*word*]

Generate possible completion matches for *word* according to the *option*s, which may be any option accepted by the **complete** builtin with the exception of **−p** and **−r**, and write the matches to the standard output. When using the **−F** or **−C** options, the various shell variables set by the programmable completion facilities, while available, will not have useful values.

The matches will be generated in the same way as if the programmable completion code had generated them directly from a completion specification with the same flags. If *word* is specified, only those completions matching *word* will be displayed.

The return value is true unless an invalid option is supplied, or no matches were generated.

complete [**−abcdefgjksuv**] [**−o** *comp-option*] [**−DE**] [**−A** *action*] [**−G** *globpat*] [**−W** *wordlist*] [**−F** *function*] [**−C** *command*]

[**−X** *filterpat*] [**−P** *prefix*] [**−S** *suffix*] *name* [*name ...*]

complete −pr [**−DE**] [*name* ...]

Specify how arguments to each *name* should be completed. If the **−p** option is supplied, or if no options are supplied, existing completion specifications are printed in a way that allows them to be reused as input. The **−r** option removes a completion specification for each *name*, or, if no *name*s are supplied, all completion specifications. The **−D** option indicates that the remaining options and actions should apply to the ‘‘default’’ command completion; that is, completion attempted on a command for which no completion has previously been defined. The **−E** option indicates that the remaining options and actions should apply to ‘‘empty’’ command completion; that is, completion attempted on a blank line.

The process of applying these completion specifications when word completion is attempted is described above under **Programmable Completion**.

Other options, if specified, have the following meanings. The arguments to the **−G**, **−W**, and **−X** options (and, if necessary, the **−P** and **−S** options) should be quoted to protect them from expansion before the **complete** builtin is invoked.

−o *comp-option*

The *comp-option* controls several aspects of the compspec’s behavior beyond the simple generation of completions. *comp-option* may be one of:

bashdefault
Perform the rest of the default **bash** completions if the compspec generates no matches.

default Use readline's default filename completion if the compspec generates no matches.

dirnames
Perform directory name completion if the compspec generates no matches.

filenames
Tell readline that the compspec generates filenames, so it can perform any file-name−specific processing (like adding a slash to directory names, quoting special characters, or suppressing trailing spaces). Intended to be used with shell functions.

noquote Tell readline not to quote the completed words if they are filenames (quoting filenames is the default).

nospace Tell readline not to append a space (the default) to words completed at the end of the line.

plusdirs After any matches defined by the compspec are generated, directory name completion is attempted and any matches are added to the results of the other actions.

−A *action*
The *action* may be one of the following to generate a list of possible completions:

alias Alias names. May also be specified as **−a**.

arrayvar
Array variable names.

binding **Readline** key binding names.

builtin Names of shell builtin commands. May also be specified as **−b**.

command
Command names. May also be specified as **−c**.

directory
Directory names. May also be specified as **−d**.

disabled
Names of disabled shell builtins.

enabled Names of enabled shell builtins.

export Names of exported shell variables. May also be specified as **−e**.

file File names. May also be specified as **−f**.

function
Names of shell functions.

group Group names. May also be specified as **−g**.

helptopic
Help topics as accepted by the **help** builtin.

hostname
Hostnames, as taken from the file specified by the **HOSTFILE** shell variable.

job Job names, if job control is active. May also be specified as **−j**.

keyword
Shell reserved words. May also be specified as **−k**.

running Names of running jobs, if job control is active.

service Service names. May also be specified as **−s**.

setopt Valid arguments for the **−o** option to the **set** builtin.

shopt Shell option names as accepted by the **shopt** builtin.

signal Signal names.

stopped Names of stopped jobs, if job control is active.

user User names. May also be specified as **−u**.

variable Names of all shell variables. May also be specified as **−v**.

−C *command*
command is executed in a subshell environment, and its output is used as the possible completions.

−F *function*
The shell function *function* is executed in the current shell environment. When the function is executed, the first argument (**$1**) is the name of the command whose arguments are being completed, the second argument (**$2**) is the word being completed, and the third argument (**$3**) is the word preceding the word being completed on the current command line. When it finishes, the possible completions are retrieved from the value of the **COMPREPLY** array variable.

−G *globpat*
The pathname expansion pattern *globpat* is expanded to generate the possible completions.

−P *prefix*
prefix is added at the beginning of each possible completion after all other options have been applied.

−S *suffix* *suffix* is appended to each possible completion after all other options have been applied.

−W *wordlist*
The *wordlist* is split using the characters in the **IFS** special variable as delimiters, and each resultant word is expanded. The possible completions are the members of the resultant list which match the word being completed.

−X *filterpat*
filterpat is a pattern as used for pathname expansion. It is applied to the list of possible completions generated by the preceding options and arguments, and each completion matching *filterpat* is removed from the list. A leading **!** in *filterpat* negates the pattern; in this case, any completion not matching *filterpat* is removed.

The return value is true unless an invalid option is supplied, an option other than **−p** or **−r** is supplied without a *name* argument, an attempt is made to remove a completion specification for a *name* for which no specification exists, or an error occurs adding a completion specification.

compopt [**−o** *option*] [**−DE**] [**+o** *option*] [*name*]
Modify completion options for each *name* according to the *option*s, or for the currently-executing completion if no *name*s are supplied. If no *option*s are given, display the completion options for each *name* or the current completion. The possible values of *option* are those valid for the **complete** builtin described above. The **−D** option indicates that the remaining options should apply to the ‘‘default’’ command completion; that is, completion attempted on a command for which no completion has previously been defined. The **−E** option indicates that the remaining options should apply to ‘‘empty’’ command completion; that is, completion attempted on a blank line.

The return value is true unless an invalid option is supplied, an attempt is made to modify the options for a *name* for which no completion specification exists, or an output error occurs.

continue [*n*]
Resume the next iteration of the enclosing **for**, **while**, **until**, or **select** 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 *n* is not greater than or equal to 1.

declare [**−aAfFgilnrtux**] [**−p**] [*name*[=*value*] ...]
typeset [**−aAfFgilnrtux**] [**−p**] [*name*[=*value*] ...]
Declare variables and/or give them attributes. If no *name*s are given then display the values of variables. The **−p** option will display the attributes and values of each *name*. When **−p** is used with *name* arguments, additional options are ignored. When **−p** is supplied without *name* arguments, it will display the attributes and values of all variables having the attributes specified by the additional options. If no other options are supplied with **−p**, **declare** will display the attributes and values of all shell variables. The **−f** option will restrict the display to shell functions. The **−F**

option inhibits the display of function definitions; only the function name and attributes are printed. If the **extdebug** shell option is enabled using **shopt**, the source file name and line number where the function is defined are displayed as well. The **−F** option implies **−f**. The **−g** option forces variables to be created or modified at the global scope, even when **declare** is executed in a shell function. It is ignored in all other cases. The following options can be used to restrict output to variables with the specified attribute or to give variables attributes:

−a Each *name* is an indexed array variable (see **Arrays** above).

−A Each *name* is an associative array variable (see **Arrays** above).

−f Use function names only.

−i The variable is treated as an integer; arithmetic evaluation (see **ARITHMETIC EVALUATION** above) is performed when the variable is assigned a value.

−l When the variable is assigned a value, all upper-case characters are converted to lower-case. The upper-case attribute is disabled.

−n Give each *name* the *nameref* attribute, making it a name reference to another variable. That other variable is defined by the value of *name*. All references and assignments to *name*, except for changing the **−n** attribute itself, are performed on the variable referenced by *name*'s value. The **−n** attribute cannot be applied to array variables.

−r Make *name*s readonly. These names cannot then be assigned values by subsequent assignment statements or unset.

−t Give each *name* the *trace* attribute. Traced functions inherit the **DEBUG** and **RETURN** traps from the calling shell. The trace attribute has no special meaning for variables.

−u When the variable is assigned a value, all lower-case characters are converted to upper-case. The lower-case attribute is disabled.

−x Mark *name*s for export to subsequent commands via the environment.

Using '+' instead of '−' turns off the attribute instead, with the exceptions that **+a** may not be used to destroy an array variable and **+r** will not remove the readonly attribute. When used in a function, **declare** and **typeset** make each *name* local, as with the **local** command, unless the **−g** option is supplied. If a variable name is followed by =*value*, the value of the variable is set to *value*. The return value is 0 unless an invalid option is encountered, an attempt is made to define a function using `-f foo=bar`, an attempt is made to assign a value to a readonly variable, an attempt is made to assign a value to an array variable without using the compound assignment syntax (see **Arrays** above), one of the *names* is not a valid shell variable name, an attempt is made to turn off readonly status for a readonly variable, an attempt is made to turn off array status for an array variable, or an attempt is made to display a non-existent function with **−f**.

dirs [−clpv] [+*n*] [−*n*]

Without options, displays the list of currently remembered directories. The default display is on a single line with directory names separated by spaces. Directories are added to the list with the **pushd** command; the **popd** command removes entries from the list.

−c Clears the directory stack by deleting all of the entries.

−l Produces a listing using full pathnames; the default listing format uses a tilde to denote the home directory.

−p Print the directory stack with one entry per line.

−v Print the directory stack with one entry per line, prefixing each entry with its index in the stack.

+*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.

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

disown [**−ar**] [**−h**] [*jobspec* ...]

Without options, remove each *jobspec* from the table of active jobs. If *jobspec* is not present, and neither **−a** nor **−r** is supplied, the shell's notion of the *current job* is used. If the **−h** option is given, each *jobspec* is not removed from the table, but is marked so that **SIGHUP** is not sent to the job if the shell receives a **SIGHUP**. If no *jobspec* is present, and neither the **−a** nor the **−r** option is supplied, the *current job* is used. If no *jobspec* is supplied, the **−a** option means to remove or mark all jobs; the **−r** option without a *jobspec* argument restricts operation to running jobs. The return value is 0 unless a *jobspec* does not specify a valid job.

echo [**−neE**] [*arg* ...]

Output the *arg*s, separated by spaces, followed by a newline. The return status is 0 unless a write error occurs. 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. The **xpg_echo** shell option may be used to dynamically determine whether or not **echo** expands these escape characters by default. **echo** does not interpret **−−** to mean the end of options. **echo** interprets the following escape sequences:

\a alert (bell)
\b backspace
\c suppress further output
\e
\E an escape character
\f form feed
\n new line
\r carriage return
\t horizontal tab
\v vertical tab
**** backslash
\0*nnn* the eight-bit character whose value is the octal value *nnn* (zero to three octal digits)
\x*HH* the eight-bit character whose value is the hexadecimal value *HH* (one or two hex digits)
\u*HHHH*
the Unicode (ISO/IEC 10646) character whose value is the hexadecimal value *HHHH* (one to four hex digits)
\U*HHHHHHHH*
the Unicode (ISO/IEC 10646) character whose value is the hexadecimal value *HHHHHHHH* (one to eight hex digits)

enable [**−a**] [**−dnps**] [**−f** *filename*] [*name* ...]

Enable and disable builtin shell commands. Disabling a builtin allows a disk command which has the same name as a shell builtin to be executed without specifying a full pathname, even though the shell normally searches for builtins before disk commands. 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, run `enable -n test`. The **−f** option means to load the new builtin command *name* from shared object *filename*, on systems that support dynamic loading. The **−d** option will delete a builtin previously loaded with **−f**. If no *name* arguments are given, or if the **−p** option is supplied, a list of shell builtins is printed. With no other option arguments, the list consists of all enabled shell builtins. If **−n** is supplied, only disabled builtins are printed. If **−a** is supplied, the list printed includes all builtins, with an indication of whether or not each is enabled. If **−s** is supplied, the output is restricted to the POSIX *special* builtins. The return value is 0 unless a *name* is not a shell builtin or there is an error loading a new builtin from a shared object.

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 **eval**. If there are no *args*, or only null arguments, **eval** returns 0.

exec [**−cl**] [**−a** *name*] [*command* [*arguments*]]

If *command* is specified, it replaces the shell. No new process is created. The *arguments* become the arguments to *command*. If the **−l** option is supplied, the shell places a dash at the beginning of the zeroth argument passed to *command*. This is what *login*(1) does. The **−c** option causes *command* to be executed with an empty environment. If **−a** is supplied, the shell passes *name* as the zeroth argument to the executed command. If *command* cannot be executed for some reason, a non-interactive shell exits, unless the **execfail** shell option is enabled. In that 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. If there is a redirection error, the return status is 1.

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 [**−fn**] [*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 names of all exported variables is printed. The **−n** option causes the export property to be removed from each *name*. If a variable name is followed by =*word*, the value of the variable is set to *word*. **export** returns an exit status of 0 unless an invalid option is encountered, one of the *names* is not a valid shell variable name, or **−f** is supplied with a *name* that is not a function.

fc [**−e** *ename*] [**−lnr**] [*first*] [*last*]
fc −s [*pat*=*rep*] [*cmd*]
The first form selects a range of commands from *first* to *last* from the history list and displays or edits and re-executes them. *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** option suppresses the command numbers when listing. The **−r** option reverses the order of the commands. If the **−l** option 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, *vi* 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*. *Command* is intepreted the same as *first* above. 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 invalid 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*]
Resume *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 characters to be recognized; if a character is followed by a colon, the option is expected to have an

argument, which should be separated from it by white space. The colon and question mark characters may not be used as option characters. 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.

When the end of options is encountered, **getopts** exits with a return value greater than zero. **OPTIND** is set to the index of the first non-option argument, and *name* is set to ?.

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

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 invalid options or missing option arguments are encountered. If the variable **OPTERR** is set to 0, no error messages will be displayed, even if the first character of *optstring* is not a colon.

If an invalid 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 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 [**−lr**] [**−p** *filename*] [**−dt**] [*name*]

Each time **hash** is invoked, the full pathname of the command *name* is determined by searching the directories in **$PATH** and remembered. Any previously-remembered pathname is discarded. If the **−p** option is supplied, no path search is performed, and *filename* is used as the full filename of the command. The **−r** option causes the shell to forget all remembered locations. The **−d** option causes the shell to forget the remembered location of each *name*. If the **−t** option is supplied, the full pathname to which each *name* corresponds is printed. If multiple *name* arguments are supplied with **−t**, the *name* is printed before the hashed full pathname. The **−l** option causes output to be displayed in a format that may be reused as input. If no arguments are given, or if only **−l** is supplied, information about remembered commands is printed. The return status is true unless a *name* is not found or an invalid option is supplied.

help [**−dms**] [*pattern*]

Display helpful information about builtin commands. If *pattern* is specified, **help** gives detailed help on all commands matching *pattern*; otherwise help for all the builtins and shell control structures is printed.

- **−d** Display a short description of each *pattern*
- **−m** Display the description of each *pattern* in a manpage-like format
- **−s** Display only a short usage synopsis for each *pattern*

The return status is 0 unless no command matches *pattern*.

history [*n*]
history −c

history −d *offset*
history −anrw [*filename*]
history −p *arg* [*arg ...*]
history −s *arg* [*arg ...*]

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 the shell variable **HISTTIMEFORMAT** is set and not null, it is used as a format string for *strftime*(3) to display the time stamp associated with each displayed history entry. No intervening blank is printed between the formatted time stamp and the history line. If *filename* 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:

−c Clear the history list by deleting all the entries.

−d *offset*
Delete the history entry at position *offset*.

−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 append them to the current history list.

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

−p Perform history substitution on the following *args* and display the result on the standard output. Does not store the results in the history list. Each *arg* must be quoted to disable normal history expansion.

−s Store the *args* in the history list as a single entry. The last command in the history list is removed before the *args* are added.

If the **HISTTIMEFORMAT** variable is set, the time stamp information associated with each history entry is written to the history file, marked with the history comment character. When the history file is read, lines beginning with the history comment character followed immediately by a digit are interpreted as timestamps for the previous history line. The return value is 0 unless an invalid option is encountered, an error occurs while reading or writing the history file, an invalid *offset* is supplied as an argument to **−d**, or the history expansion supplied as an argument to **−p** fails.

jobs [**−lnprs**] [*jobspec* ...]
jobs −x *command* [*args* ...]

The first form lists the active jobs. The options have the following meanings:

−l List process IDs in addition to the normal information.

−n Display information only about jobs that have changed status since the user was last notified of their status.

−p List only the process ID of the job's process group leader.

−r Display only running jobs.

−s Display only stopped jobs.

If *jobspec* is given, output is restricted to information about that job. The return status is 0 unless an invalid option is encountered or an invalid *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* | **−n** *signum* | **−***sigspec*] [*pid* | *jobspec*] ...
kill −l [*sigspec* | *exit_status*]

Send the signal named by *sigspec* or *signum* to the processes named by *pid* or *jobspec*. *sigspec* is either a case-insensitive signal name such as **SIGKILL** (with or without the **SIG** prefix) or a signal number; *signum* is a signal number. 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 signals corresponding to the arguments are listed, and the return status is 0. The *exit_status* argument to **−l** is a number specifying either a signal number or the exit status of a process terminated by a signal. **kill** returns true if at least one signal was successfully sent, or false if an error occurs or an invalid option is encountered.

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

local [*option*] [*name*[=*value*] ...]
For each argument, a local variable named *name* is created, and assigned *value*. The *option* can be any of the options accepted by **declare**. 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, an invalid *name* is supplied, or *name* is a readonly variable.

logout Exit a login shell.

mapfile [**−n** *count*] [**−O** *origin*] [**−s** *count*] [**−t**] [**−u** *fd*] [**−C** *callback*] [**−c** *quantum*] [*array*]
readarray [**−n** *count*] [**−O** *origin*] [**−s** *count*] [**−t**] [**−u** *fd*] [**−C** *callback*] [**−c** *quantum*] [*array*]
Read lines from the standard input into the indexed array variable *array*, or from file descriptor *fd* if the **−u** option is supplied. The variable **MAPFILE** is the default *array*. Options, if supplied, have the following meanings:

- **−n** Copy at most *count* lines. If *count* is 0, all lines are copied.
- **−O** Begin assigning to *array* at index *origin*. The default index is 0.
- **−s** Discard the first *count* lines read.
- **−t** Remove a trailing newline from each line read.
- **−u** Read lines from file descriptor *fd* instead of the standard input.
- **−C** Evaluate *callback* each time *quantum* lines are read. The **−c** option specifies *quantum*.
- **−c** Specify the number of lines read between each call to *callback*.

If **−C** is specified without **−c**, the default quantum is 5000. When *callback* is evaluated, it is supplied the index of the next array element to be assigned and the line to be assigned to that element as additional arguments. *callback* is evaluated after the line is read but before the array element is assigned.

If not supplied with an explicit origin, **mapfile** will clear *array* before assigning to it.

mapfile returns successfully unless an invalid option or option argument is supplied, *array* is invalid or unassignable, or if *array* is not an indexed array.

popd [**−n**] [+*n*] [−*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. Arguments, if supplied, have the following meanings:

- **−n** Suppresses the normal change of directory when removing directories from the stack, so that only the stack is manipulated.
- **+***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 invalid option is encountered, the directory stack is empty, a non-existent directory stack entry is specified, or the directory change fails.

printf [**−v** *var*] *format* [*arguments*]
Write the formatted *arguments* to the standard output under the control of the *format*. The **−v** option causes the output to be assigned to the variable *var* rather than being printed to the standard output.

The *format* is a character string which contains three types of objects: plain characters, which are simply copied to standard output, character escape sequences, which are converted and copied to the standard output, and format specifications, each of which causes printing of the next successive *argument*. In addition to the standard *printf*(1) format specifications, **printf** interprets the following extensions:

%b causes **printf** to expand backslash escape sequences in the corresponding *argument* (except that **\c** terminates output, backslashes in **\'**, **\"**, and **\?** are not removed, and octal escapes beginning with **\0** may contain up to four digits).

%q causes **printf** to output the corresponding *argument* in a format that can be reused as shell input.

%(*datefmt***)T**
causes **printf** to output the date-time string resulting from using *datefmt* as a format string for *strftime*(3). The corresponding *argument* is an integer representing the number of seconds since the epoch. Two special argument values may be used: -1 represents the current time, and -2 represents the time the shell was invoked. If no argument is specified, conversion behaves as if -1 had been given. This is an exception to the usual **printf** behavior.

Arguments to non-string format specifiers are treated as C constants, except that a leading plus or minus sign is allowed, and if the leading character is a single or double quote, the value is the ASCII value of the following character.

The *format* is reused as necessary to consume all of the *arguments*. If the *format* requires more *arguments* than are supplied, the extra format specifications behave as if a zero value or null string, as appropriate, had been supplied. The return value is zero on success, non-zero on failure.

pushd [**−n**] [+*n*] [−*n*]
pushd [**−n**] [*dir*]
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. Arguments, if supplied, have the following meanings:

−n Suppresses the normal change of directory when adding directories to the stack, so that only the stack is manipulated.

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

−*n* Rotates the stack so that the *n*th directory (counting from the right of the list shown by **dirs**, starting with zero) is at the top.

dir Adds *dir* to the directory stack at the top, making it the new current working directory as if it had been supplied as the argument to the **cd** builtin.

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-existent directory stack element is specified, or the directory change to the specified new current directory fails.

pwd [**−LP**]
Print the absolute pathname of the current working directory. The pathname printed contains no symbolic links if the **−P** option is supplied or the **−o physical** option to the **set** builtin command is enabled. If the **−L** option is used, the pathname printed may contain symbolic links. The return status is 0 unless an error occurs while reading the name of the current directory or an invalid option is supplied.

read [**−ers**] [**−a** *aname*] [**−d** *delim*] [**−i** *text*] [**−n** *nchars*] [**−N** *nchars*] [**−p** *prompt*] [**−t** *timeout*] [**−u** *fd*] [*name* ...]
One line is read from the standard input, or from the file descriptor *fd* supplied as an argument to the **−u** option, and the first word is assigned to the first *name*, the second word to the second *name*, and so on, with leftover words and their intervening separators assigned to the last *name*. If there

are fewer words read from the input stream than names, the remaining names are assigned empty values. The characters in **IFS** are used to split the line into words. The backslash character (\) may be used to remove any special meaning for the next character read and for line continuation. Options, if supplied, have the following meanings:

−a *aname*
: The words are assigned to sequential indices of the array variable *aname*, starting at 0. *aname* is unset before any new values are assigned. Other *name* arguments are ignored.

−d *delim*
: The first character of *delim* is used to terminate the input line, rather than newline.

−e
: If the standard input is coming from a terminal, **readline** (see **READLINE** above) is used to obtain the line. Readline uses the current (or default, if line editing was not previously active) editing settings.

−i *text*
: If **readline** is being used to read the line, *text* is placed into the editing buffer before editing begins.

−n *nchars*
: **read** returns after reading *nchars* characters rather than waiting for a complete line of input, but honor a delimiter if fewer than *nchars* characters are read before the delimiter.

−N *nchars*
: **read** returns after reading exactly *nchars* characters rather than waiting for a complete line of input, unless EOF is encountered or **read** times out. Delimiter characters encountered in the input are not treated specially and do not cause **read** to return until *nchars* characters are read.

−p *prompt*
: Display *prompt* on standard error, without a trailing newline, before attempting to read any input. The prompt is displayed only if input is coming from a terminal.

−r
: Backslash does not act as an escape character. The backslash is considered to be part of the line. In particular, a backslash-newline pair may not be used as a line continuation.

−s
: Silent mode. If input is coming from a terminal, characters are not echoed.

−t *timeout*
: Cause **read** to time out and return failure if a complete line of input (or a specified number of characters) is not read within *timeout* seconds. *timeout* may be a decimal number with a fractional portion following the decimal point. This option is only effective if **read** is reading input from a terminal, pipe, or other special file; it has no effect when reading from regular files. If **read** times out, **read** saves any partial input read into the specified variable *name*. If *timeout* is 0, **read** returns immediately, without trying to read any data. The exit status is 0 if input is available on the specified file descriptor, non-zero otherwise. The exit status is greater than 128 if the timeout is exceeded.

−u *fd*
: Read input from file descriptor *fd*.

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, **read** times out (in which case the return code is greater than 128), a variable assignment error (such as assigning to a readonly variable) occurs, or an invalid file descriptor is supplied as the argument to **−u**.

readonly [**−aAf**] [**−p**] [*name*[=*word*] ...]
: The given *names* are marked readonly; 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. The **−a** option restricts the variables to indexed arrays; the **−A** option restricts the variables to associative arrays. If both options are supplied, **−A** takes precedence. If no *name* arguments are given, or if the **−p** option is supplied, a list of all readonly names is printed. The other options may be used to restrict the output to a subset of the set of readonly names. The **−p** option causes output to be displayed in a format that may be reused as input. If a variable name is followed by =*word*, the value of the variable is set to *word*. The return status is 0 unless an invalid option is encountered, one of the *names* is not a valid shell variable name, or **−f** is supplied with a *name* that is not a function.

return [*n*]

Causes a function to stop executing and return the value specified by *n* to its caller. If *n* is omitted, the return status is that of the last command executed in the function body. If **return** is 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 *n* is supplied, the return value is its least significant 8 bits. The return status is non-zero if **return** is supplied a non-numeric argument, or is used outside a function and not during execution of a script by **.** or **source**. Any command associated with the **RETURN** trap is executed before execution resumes after the function or script.

set [**−−abefhkmnptuvxBCEHPT**] [**−o** *option−name*] [*arg* ...]
set [**+abefhkmnptuvxBCEHPT**] [**+o** *option−name*] [*arg* ...]

Without options, the name and value of each shell variable are displayed in a format that can be reused as input for setting or resetting the currently-set variables. Read-only variables cannot be reset. In *posix mode*, only shell variables are listed. The output is sorted according to the current locale. When options are specified, they set or unset shell attributes. Any arguments remaining after option processing are treated as values for the positional parameters and are assigned, in order, to **$1**, **$2**, **...** **$***n*. Options, if specified, have the following meanings:

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

−b Report the status of terminated background jobs immediately, rather than before the next primary prompt. This is effective only when job control is enabled.

−e Exit immediately if a *pipeline* (which may consist of a single *simple command*), a *list*, or a *compound command* (see **SHELL GRAMMAR** above), exits with a non-zero status. The shell does not exit if the command that fails is part of the command list immediately following a **while** or **until** keyword, part of the test following the **if** or **elif** reserved words, part of any command executed in a **&&** or **||** list except the command following the final **&&** or **||**, any command in a pipeline but the last, or if the command's return value is being inverted with **!**. If a compound command other than a subshell returns a non-zero status because a command failed while **−e** was being ignored, the shell does not exit. A trap on **ERR**, if set, is executed before the shell exits. This option applies to the shell environment and each subshell environment separately (see **COMMAND EXECUTION ENVIRONMENT** above), and may cause subshells to exit before executing all the commands in the subshell.

If a compound command or shell function executes in a context where **−e** is being ignored, none of the commands executed within the compound command or function body will be affected by the **−e** setting, even if **−e** is set and a command returns a failure status. If a compound command or shell function sets **−e** while executing in a context where **−e** is ignored, that setting will not have any effect until the compound command or the command containing the function call completes.

−f Disable pathname expansion.

−h Remember the location of commands as they are looked up for execution. This is enabled by default.

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

−m Monitor mode. Job control is enabled. This option is on by default for interactive shells on systems that support it (see **JOB CONTROL** above). All processes run in a separate process group. When a background job completes, the shell prints a line containing its exit status.

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

−o *option−name*

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

allexport
Same as **−a**.
braceexpand
Same as **−B**.
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 **−−noediting** option. This also affects the editing interface used for **read −e**.
errexit Same as **−e**.
errtrace Same as **−E**.
functrace
Same as **−T**.
hashall Same as **−h**.
histexpand
Same as **−H**.
history Enable command history, as described above under **HISTORY**. This option is on by default in interactive shells.
ignoreeof
The effect is as if the shell command `IGNOREEOF=10` had been executed (see **Shell Variables** above).
keyword
Same as **−k**.
monitor Same as **−m**.
noclobber
Same as **−C**.
noexec Same as **−n**.
noglob Same as **−f**.
nolog Currently ignored.
notify Same as **−b**.
nounset Same as **−u**.
onecmd Same as **−t**.
physical Same as **−P**.
pipefail If set, the return value of a pipeline is the value of the last (rightmost) command to exit with a non-zero status, or zero if all commands in the pipeline exit successfully. This option is disabled by default.
posix Change the behavior of **bash** where the default operation differs from the POSIX standard to match the standard (*posix mode*).
privileged
Same as **−p**.
verbose Same as **−v**.
vi Use a vi-style command line editing interface. This also affects the editing interface used for **read −e**.
xtrace Same as **−x**.

If **−o** is supplied with no *option−name*, the values of the current options are printed. If **+o** is supplied with no *option−name*, a series of **set** commands to recreate the current option settings is displayed on the standard output.

−p Turn on *privileged* mode. In this mode, the **$ENV** and **$BASH_ENV** files are not processed, shell functions are not inherited from the environment, and the **SHELLOPTS**, **BASHOPTS**, **CDPATH**, and **GLOBIGNORE** variables, if they appear in the environment, are ignored. If the shell is started with the effective user (group) id not equal to the real user (group) id, and the **−p** option is not supplied, these actions are taken and the effective user id is set to the real user id. If the **−p** option is supplied at startup, the effective user id is not reset. 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 and parameters other than the special parameters "@" and "*" as an error when performing parameter expansion. If expansion is attempted on an unset variable or parameter, 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*, **for** command, **case** command, **select** command, or arithmetic **for** command, display the expanded value of **PS4**, followed by the command and its expanded arguments or associated word list.

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

−C If set, **bash** does not overwrite an existing file with the **>**, **>&**, and **<>** redirection operators. This may be overridden when creating output files by using the redirection operator **>|** instead of **>**.

−E If set, any trap on **ERR** is inherited by shell functions, command substitutions, and commands executed in a subshell environment. The **ERR** trap is normally not inherited in such cases.

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

−P If set, the shell does not resolve symbolic links when executing commands such as **cd** that change the current working directory. It uses the physical directory structure instead. By default, **bash** follows the logical chain of directories when performing commands which change the current directory.

−T If set, any traps on **DEBUG** and **RETURN** are inherited by shell functions, command substitutions, and commands executed in a subshell environment. The **DEBUG** and **RETURN** traps are normally not inherited in such cases.

−− If no arguments follow this option, 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 options are off by default unless otherwise noted. Using + rather than − causes these options to be turned off. The options can also be specified as arguments to an invocation of the shell. The current set of options may be found in **$−**. The return status is always true unless an invalid option is encountered.

shift [*n*]

The positional parameters from *n*+1 ... are renamed to **$1 ....** Parameters represented by the numbers **$#** down to **$#**−*n*+1 are unset. *n* must be a non-negative number less than or equal to **$#**. If *n* is 0, no parameters are changed. If *n* is not given, it is assumed to be 1. If *n* is greater than **$#**, the positional parameters are not changed. The return status is greater than zero if *n* is greater than **$#** or less than zero; otherwise 0.

shopt [**−pqsu**] [**−o**] [*optname* ...]

Toggle the values of variables controlling optional shell behavior. With no options, or with the **−p** option, a list of all settable options is displayed, with an indication of whether or not each is set. The **−p** option causes output to be displayed in a form that may be reused as input. Other options have the following meanings:

−s Enable (set) each *optname*.

−u Disable (unset) each *optname*.

−q Suppresses normal output (quiet mode); the return status indicates whether the *optname* is set or unset. If multiple *optname* arguments are given with **−q**, the return status is zero if all *optnames* are enabled; non-zero otherwise.

−o Restricts the values of *optname* to be those defined for the **−o** option to the **set** builtin.

If either **−s** or **−u** is used with no *optname* arguments, **shopt** shows only those options which are set or unset, respectively. Unless otherwise noted, the **shopt** options are disabled (unset) by default.

The return status when listing options is zero if all *optnames* are enabled, non-zero otherwise. When setting or unsetting options, the return status is zero unless an *optname* is not a valid shell option.

The list of **shopt** options is:

autocd If set, a command name that is the name of a directory is executed as if it were the argument to the **cd** command. This option is only used by interactive shells.

cdable_vars
If set, an argument to the **cd** builtin command that is not a directory is assumed to be the name of a variable whose value is the directory to change to.

cdspell If set, minor errors in the spelling of a directory component in a **cd** command will be corrected. The errors checked for are transposed characters, a missing character, and one character too many. If a correction is found, the corrected filename is printed, and the command proceeds. This option is only used by interactive shells.

checkhash
If set, **bash** checks that a command found in the hash table exists before trying to execute it. If a hashed command no longer exists, a normal path search is performed.

checkjobs
If set, **bash** lists the status of any stopped and running jobs before exiting an interactive shell. If any jobs are running, this causes the exit to be deferred until a second exit is attempted without an intervening command (see **JOB CONTROL** above). The shell always postpones exiting if any jobs are stopped.

checkwinsize
If set, **bash** checks the window size after each command and, if necessary, updates the values of **LINES** and **COLUMNS**.

cmdhist If set, **bash** attempts to save all lines of a multiple-line command in the same history entry. This allows easy re-editing of multi-line commands.

compat31
If set, **bash** changes its behavior to that of version 3.1 with respect to quoted arguments to the **[[** conditional command's **=~** operator and locale-specific string comparison when using the **[[** conditional command's **<** and **>** operators. Bash versions prior to bash-4.1 use ASCII collation and *strcmp*(3); bash-4.1 and later use the current locale's collation sequence and *strcoll*(3).

compat32
If set, **bash** changes its behavior to that of version 3.2 with respect to locale-specific string comparison when using the **[[** conditional command's **<** and **>** operators (see previous item).

compat40
If set, **bash** changes its behavior to that of version 4.0 with respect to locale-specific string comparison when using the **[[** conditional command's **<** and **>** operators (see description of **compat31**) and the effect of interrupting a command list. Bash versions 4.0 and later interrupt the list as if the shell received the interrupt; previous versions continue with the next command in the list.

compat41
If set, **bash**, when in posix mode, treats a single quote in a double-quoted parameter expansion as a special character. The single quotes must match (an even number) and the characters between the single quotes are considered quoted. This is the behavior of posix mode through version 4.1. The default bash behavior remains as in previous versions.

compat42
If set, **bash** does not process the replacement string in the pattern substitution word expansion using quote removal.

complete_fullquote
If set, **bash** quotes all shell metacharacters in filenames and directory names when performing completion. If not set, **bash** removes metacharacters such as the dollar sign from the set of characters that will be quoted in completed filenames when these metacharacters appear in shell variable references in words to be completed. This means that dollar signs in variable names that expand to directories will not be quoted; however, any dollar signs appearing in filenames will not be quoted, either. This is active only when bash is using backslashes to quote completed filenames. This variable is set by default, which is the default bash behavior in versions through 4.2.

direxpand
If set, **bash** replaces directory names with the results of word expansion when performing filename completion. This changes the contents of the readline editing buffer. If not set, **bash** attempts to preserve what the user typed.

dirspell If set, **bash** attempts spelling correction on directory names during word completion if the directory name initially supplied does not exist.

dotglob If set, **bash** includes filenames beginning with a '.' in the results of pathname expansion.

execfail If set, a non-interactive shell will not exit if it cannot execute the file specified as an argument to the **exec** builtin command. An interactive shell does not exit if **exec** fails.

expand_aliases
If set, aliases are expanded as described above under **ALIASES**. This option is enabled by default for interactive shells.

extdebug
If set, behavior intended for use by debuggers is enabled:

1. The **−F** option to the **declare** builtin displays the source file name and line number corresponding to each function name supplied as an argument.

2. If the command run by the **DEBUG** trap returns a non-zero value, the next command is skipped and not executed.

3. If the command run by the **DEBUG** trap returns a value of 2, and the shell is executing in a subroutine (a shell function or a shell script executed by the **.** or **source** builtins), a call to **return** is simulated.

4. **BASH_ARGC** and **BASH_ARGV** are updated as described in their descriptions above.

5. Function tracing is enabled: command substitution, shell functions, and subshells invoked with **(** *command* **)** inherit the **DEBUG** and **RETURN** traps.

6. Error tracing is enabled: command substitution, shell functions, and subshells invoked with **(** *command* **)** inherit the **ERR** trap.

extglob If set, the extended pattern matching features described above under **Pathname Expansion** are enabled.

extquote
If set, **$**'*string*' and **$**"*string*" quoting is performed within **${***parameter***}** expansions enclosed in double quotes. This option is enabled by default.

failglob If set, patterns which fail to match filenames during pathname expansion result in an expansion error.

force_fignore
If set, the suffixes specified by the **FIGNORE** shell variable cause words to be ignored when performing word completion even if the ignored words are the only possible completions. See **SHELL VARIABLES** above for a description of **FIGNORE**. This option is enabled by default.

globasciiranges
If set, range expressions used in pattern matching (see **Pattern Matching** above) behave as if in the traditional C locale when performing comparisons. That is, the current locale's collating sequence is not taken into account, so **b** will not collate between **A** and **B**, and upper-case and lower-case ASCII characters will collate together.

globstar If set, the pattern ** used in a pathname expansion context will match all files and zero or more directories and subdirectories. If the pattern is followed by a **/**, only directories and subdirectories match.

gnu_errfmt
If set, shell error messages are written in the standard GNU error message format.

histappend
If set, the history list is appended to the file named by the value of the **HISTFILE** variable when the shell exits, rather than overwriting the file.

histreedit
If set, and **readline** is being used, a user is given the opportunity to re-edit a failed history substitution.

histverify
If set, and **readline** is being used, the results of history substitution are not immediately passed to the shell parser. Instead, the resulting line is loaded into the **readline** editing buffer, allowing further modification.

hostcomplete
If set, and **readline** is being used, **bash** will attempt to perform hostname completion when a word containing a @ is being completed (see **Completing** under **READLINE** above). This is enabled by default.

huponexit
If set, **bash** will send **SIGHUP** to all jobs when an interactive login shell exits.

interactive_comments
If set, 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). This option is enabled by default.

lastpipe If set, and job control is not active, the shell runs the last command of a pipeline not executed in the background in the current shell environment.

lithist If set, and the **cmdhist** option is enabled, multi-line commands are saved to the history with embedded newlines rather than using semicolon separators where possible.

login_shell
The shell sets this option if it is started as a login shell (see **INVOCATION** above). The value may not be changed.

mailwarn
If set, and a file that **bash** is checking for mail has been accessed since the last time it was checked, the message ‘‘The mail in *mailfile* has been read’’ is displayed.

no_empty_cmd_completion
If set, and **readline** is being used, **bash** will not attempt to search the **PATH** for possible completions when completion is attempted on an empty line.

nocaseglob
If set, **bash** matches filenames in a case−insensitive fashion when performing pathname expansion (see **Pathname Expansion** above).

nocasematch
If set, **bash** matches patterns in a case−insensitive fashion when performing matching while executing **case** or **[[** conditional commands.

nullglob
If set, **bash** allows patterns which match no files (see **Pathname Expansion** above) to expand to a null string, rather than themselves.

progcomp
If set, the programmable completion facilities (see **Programmable Completion** above) are enabled. This option is enabled by default.

promptvars
If set, prompt strings undergo parameter expansion, command substitution, arithmetic expansion, and quote removal after being expanded as described in **PROMPTING** above. This option is enabled by default.

restricted_shell
The shell sets this option if it is started in restricted mode (see **RESTRICTED SHELL** below). The value may not be changed. This is not reset when the startup files are executed, allowing the startup files to discover whether or not a shell is restricted.

shift_verbose
If set, the **shift** builtin prints an error message when the shift count exceeds the number of positional parameters.

sourcepath
If set, the **source** (**.**) builtin uses the value of **PATH** to find the directory containing the file supplied as an argument. This option is enabled by default.

xpg_echo
If set, the **echo** builtin expands backslash-escape sequences by default.

suspend [**−f**]
Suspend the execution of this shell until it receives a **SIGCONT** signal. A login shell cannot be suspended; the **−f** option can be used to override this and force the suspension. 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*. Each operator and operand must be a separate argument. Expressions are composed of the primaries described above under **CONDITIONAL EXPRESSIONS**. **test** does not accept any options, nor does it accept and ignore an argument of **−−** as signifying the end of options.

Expressions may be combined using the following operators, listed in decreasing order of precedence. The evaluation depends on the number of arguments; see below. Operator precedence is used when there are five or more arguments.

! *expr* True if *expr* is false.

(*expr*) Returns the value of *expr*. This may be used to override the normal precedence of operators.

expr1 **−a** *expr2*
True if both *expr1* and *expr2* are true.

expr1 **−o** *expr2*
True if either *expr1* or *expr2* is true.

test and **[** evaluate conditional expressions using a set of rules based on the number of arguments.

0 arguments
The expression is false.

1 argument
The expression is true if and only if the argument is not null.

2 arguments
If the first argument is **!**, the expression is true if and only if the second argument is null. If the first argument is one of the unary conditional operators listed above under **CONDITIONAL EXPRESSIONS**, the expression is true if the unary test is true. If the first argument is not a valid unary conditional operator, the expression is false.

3 arguments
The following conditions are applied in the order listed. If the second argument is one of the binary conditional operators listed above under **CONDITIONAL EXPRESSIONS**, the result of the expression is the result of the binary test using the first and third arguments as operands. The **−a** and **−o** operators are considered binary operators when there are three arguments. If the first argument is **!**, the value is the negation of the two-argument test using the second and third arguments. If the first argument is exactly (and the third argument is exactly), the result is the one-argument test of the second argument. Otherwise, the expression is false.

4 arguments

If the first argument is **!**, the result is the negation of the three-argument expression composed of the remaining arguments. Otherwise, the expression is parsed and evaluated according to precedence using the rules listed above.

5 or more arguments

The expression is parsed and evaluated according to precedence using the rules listed above.

When used with **test** or **[**, the **<** and **>** operators sort lexicographically using ASCII ordering.

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

trap [**−lp**] [[*arg*] *sigspec* ...]

The command *arg* is to be read and executed when the shell receives signal(s) *sigspec*. If *arg* is absent (and there is a single *sigspec*) or **−**, each specified signal is reset to its original disposition (the value it had upon entrance to the shell). If *arg* is the null string the signal specified by each *sigspec* is ignored by the shell and by the commands it invokes. If *arg* is not present and **−p** has been supplied, then the trap commands associated with each *sigspec* are displayed. If no arguments are supplied or if only **−p** is given, **trap** prints the list of commands associated with each signal. The **−l** option causes the shell to print a list of signal names and their corresponding numbers. Each *sigspec* is either a signal name defined in <*signal.h*>, or a signal number. Signal names are case insensitive and the **SIG** prefix is optional.

If a *sigspec* is **EXIT** (0) the command *arg* is executed on exit from the shell. If a *sigspec* is **DEBUG**, the command *arg* is executed before every *simple command*, *for* command, *case* command, *select* command, every arithmetic *for* command, and before the first command executes in a shell function (see **SHELL GRAMMAR** above). Refer to the description of the **extdebug** option to the **shopt** builtin for details of its effect on the **DEBUG** trap. If a *sigspec* is **RETURN**, the command *arg* is executed each time a shell function or a script executed with the **.** or **source** builtins finishes executing.

If a *sigspec* is **ERR**, the command *arg* is executed whenever a a pipeline (which may consist of a single simple command), a list, or a compound command returns a non−zero exit status, subject to the following conditions. The **ERR** trap is not executed if the failed command is part of the command list immediately following a **while** or **until** keyword, part of the test in an *if* statement, part of a command executed in a **&&** or **||** list except the command following the final **&&** or **||**, any command in a pipeline but the last, or if the command's return value is being inverted using **!**. These are the same conditions obeyed by the **errexit** (**−e**) option.

Signals ignored upon entry to the shell cannot be trapped or reset. Trapped signals that are not being ignored are reset to their original values in a subshell or subshell environment when one is created. The return status is false if any *sigspec* is invalid; otherwise **trap** returns true.

type [**−aftpP**] *name* [*name* ...]

With no options, indicate how each *name* would be interpreted if used as a command name. If the **−t** option is used, **type** prints a string 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 **−p** option 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 -t name` would not return *file*. The **−P** option forces a **PATH** search for each *name*, even if `type -t name` would not return *file*. If a command is hashed, **−p** and **−P** print the hashed value, which is not necessarily the file that appears first in **PATH**. If the **−a** option is used, **type** prints all of the places that contain an executable named *name*. This includes aliases and functions, if and only if the **−p** option is not also used. The table of hashed commands is not consulted when using **−a**. The **−f** option suppresses shell function lookup, as with the **command** builtin. **type** returns true if all of the arguments are found, false if any are not found.

ulimit [**−HSTabcdefilmnpqrstuvx** [*limit*]]

Provides control over the resources available to the shell and to processes started by it, on systems that allow such control. The **−H** and **−S** options specify that the hard or soft limit is set for the given resource. A hard limit cannot be increased by a non-root user 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, both the soft and hard limits are set. The value of *limit* can be a number in the unit specified for the resource or one of the special values **hard**, **soft**, or **unlimited**, which stand for the current hard limit, the current soft limit, and no limit, respectively. 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 are printed before the value. Other options are interpreted as follows:

−a All current limits are reported
−b The maximum socket buffer size
−c The maximum size of core files created
−d The maximum size of a process's data segment
−e The maximum scheduling priority ("nice")
−f The maximum size of files written by the shell and its children
−i The maximum number of pending signals
−l The maximum size that may be locked into memory
−m The maximum resident set size (many systems do not honor this limit)
−n The maximum number of open file descriptors (most systems do not allow this value to be set)
−p The pipe size in 512-byte blocks (this may not be set)
−q The maximum number of bytes in POSIX message queues
−r The maximum real-time scheduling priority
−s The maximum stack size
−t The maximum amount of cpu time in seconds
−u The maximum number of processes available to a single user
−v The maximum amount of virtual memory available to the shell and, on some systems, to its children
−x The maximum number of file locks
−T The maximum number of threads

If *limit* is given, and the **−a** option is not used, *limit* is the new value of the specified resource. 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 **−T**, **−b**, **−n**, and **−u**, which are unscaled values. The return status is 0 unless an invalid option or argument is supplied, or an error occurs while setting a new limit.

umask [**−p**] [**−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, 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. If the **−p** option is supplied, and *mode* is omitted, the output is in a form that may be reused as input. 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 each *name* 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**] [−**n**] [*name* ...]

For each *name*, remove the corresponding variable or function. If the **−v** option is given, each *name* refers to a shell variable, and that variable is removed. Read-only variables may not be unset. If **−f** is specified, each *name* refers to a shell function, and the function definition is removed. If the **−n** option is supplied, and *name* is a variable with the *nameref* attribute, *name* will be unset rather than the variable it references. **−n** has no effect if the **−f** option is supplied. If no options are supplied, each *name* refers to a variable; if there is no variable by that name, any function with that name is unset. Each unset variable or function is removed from the environment passed to subsequent commands. If any of **COMP_WORDBREAKS**, **RANDOM**, **SECONDS**, **LINENO**, **HISTCMD**, **FUNCNAME**, **GROUPS**, or **DIRSTACK** are unset, they lose their special properties, even if they are subsequently reset. The exit status is true unless a *name* is readonly.

wait [**−-n**] [*n* ...]
Wait for each specified process and return its termination status. Each *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 the **−-n** option is supplied, **wait** waits for any job to terminate and returns its exit status. If *n* specifies a non-existent 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)