NAME

:, ., [, 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, readarray, 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**, **return**, **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. The return status is zero.

. [**–p** *path*] *filename* [*arguments*]
source [**–p** *path*] *filename* [*arguments*]

The **.** command (**source**) reads and execute commands from *filename* in the current shell environment and returns the exit status of the last command executed from *filename*.

If *filename* does not contain a slash, **.** searches for it. If the **–p** option is supplied, **.** treats *path* as a colon-separated list of directories in which to find *filename*; otherwise, **.** uses the entries in **PATH** to find the directory containing *filename*. *filename* does not need to be executable. When **bash** is not in posix mode, it searches the current directory if *filename* is not found in **PATH**, but does not search the current directory if **–p** is supplied. If the **sourcepath** option to the **shopt** builtin command is turned off, **.** does not search **PATH**.

If any *arguments* are supplied, they become the positional parameters when *filename* is executed. Otherwise the positional parameters are unchanged.

If the **–T** option is enabled, **.** inherits any trap on **DEBUG**; if it is not, any **DEBUG** trap string is saved and restored around the call to **.**, and **.** unsets the **DEBUG** trap while it executes. If **–T** is not set, and the sourced file changes the **DEBUG** trap, the new value persists after **.** completes. The return status is the status of the last command executed from *filename* (0 if no commands are executed), and non-zero if *filename* is not found or cannot be read.

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

With no arguments or with the **–p** option, **alias** prints the list of aliases in the form **alias** *name*=*value* on standard output. When arguments are supplied, define an alias 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 during command parsing. For each *name* in the argument list for which no *value* is supplied, print the name and value of the alias *name*. **alias** returns true unless a *name* is given (without a corresponding =*value*) 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 uses its notion of the *current job*. **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*] [**–lsvSVX**]
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 [**−m** *keymap*] **−p**|**−P** [*readline−command*]
bind [**−m** *keymap*] *keyseq*:*readline−command*
bind *readline-command-line*

Display current **readline** key and function bindings, bind a key sequence to a **readline** function or macro or to a shell command, or set a **readline** variable. Each non-option argument is a key binding or command as it would appear in a **readline** initialization file such as *.inputrc*, but each binding or command must be passed as a separate argument; e.g., \C−x\C−r: re−read−init−file. In the following descriptions, output available to be re-read is formatted as commands that would appear in a **readline** initialization file or that would be supplied as individual arguments to a **bind** command. 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* (*vi−move* is also a synonym); *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 used as an argument to a subsequent **bind** command or in a **readline** initialization file. If arguments remain after option processing, **bind** treats them as **readline** command names and restricts output to those names.

−P
: List current **readline** function names and bindings. If arguments remain after option processing, **bind** treats them as **readline** command names and restricts output to those names.

−s
: Display **readline** key sequences bound to macros and the strings they output in such a way that they can be used as an argument to a subsequent **bind** command or in a **readline** initialization file.

−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 used as an argument to a subsequent **bind** command or in a **readline** initialization file.

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

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

−q *function*
: Display key sequences that invoke the named **readline** *function*.

−u *function*
: Unbind all key sequences bound to the named **readline** *function*.

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

−x *keyseq***[:]***shell−command*
: Cause *shell−command* to be executed whenever *keyseq* is entered. The separator between *keyseq* and *shell−command* is either whitespace or a colon optionally followed by whitespace. If the separator is whitespace, *shell−command* must be enclosed in double quotes and **readline** expands any of its special backslash-escapes in *shell−command* before saving it. If the separator is a colon, any enclosing double quotes are optional, and **readline** does not expand the command string before saving it. Since the entire key binding expression must be a single argument, it should be enclosed in single quotes. When *shell−command* is executed, the shell sets the **READLINE_LINE** variable to the contents of the **readline** line buffer and the **READLINE_POINT** and **READLINE_MARK** variables to the current location of the insertion point and the saved insertion point (the mark), respectively. The shell assigns any numeric argument the user supplied to the **READLINE_ARGUMENT** variable. If there was no argument, that variable is not set. If the executed command changes the value of any of **READLINE_LINE**, **READLINE_POINT**, or **READLINE_MARK**, 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 an argument to a subsequent **bind** command.

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

break [*n*]

Exit from within a **for**, **while**, **until**, or **select** loop. If *n* is specified, **break** exits *n* enclosing loops. *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 *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**] [−@] [*dir*]
cd −P [**−e**] [−@] [*dir*]

Change the current directory to *dir*. if *dir* is not supplied, the value of the **HOME** shell variable is used as *dir*. The variable **CDPATH** exists, and *dir* does not begin with a slash (/), **cd** uses it as a search path: the shell searches each directory name in **CDPATH** 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.,

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 **cd** to follow symbolic links by resolving the link after processing instances of .. in *dir*. If .. appears in *dir*, **cd** processes it by removing the immediately previous pathname component from *dir*, back to a slash or the beginning of *dir*, and verifying that the portion of *dir* it has processed to that point is still a valid directory name after removing the pathname component. If it is not a valid directory name, **cd** returns a non-zero status. If neither **−L** nor **−P** is supplied, **cd** behaves as if **−L** had been supplied.

If the **−e** option is supplied with **−P**, and **cd** cannot successfully determine the current working directory after a successful directory change, it will return a non-zero status.

On systems that support it, the **−@** option presents the extended attributes associated with a file as a directory.

An argument of **−** is converted to **$OLDPWD** before attempting the directory change.

If **cd** uses a non-empty directory name from **CDPATH,** or if **−** is the first argument, and the directory change is successful, **cd** writes the absolute pathname of the new working directory to the standard output.

If the directory change is successful, **cd** sets the value of the **PWD** environment variable to the new directory name, and sets the **OLDPWD** environment variable to the value of the current working directory before the change.

The return value is true if the directory was successfully changed; false otherwise.

command [**−pVv**] *command* [*arg* ...]
The **command** builtin runs *command* with *args* suppressing the normal shell function lookup for *command*. Only builtin commands or commands found in the **PATH** named *command* are executed. If the **−p** option is supplied, 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, **command** prints a description of *command*. The **−v** option displays a single word indicating the command or filename used to invoke *command*; the **−V** option produces a more verbose description.

If the **−V** or **−v** option is supplied, the exit status is zero if *command* was found, and non-zero 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 [**−V** *varname*] [*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 exceptions of **−p**, **−r**, **−D**, **−E**, and **−I**, and write the matches to the standard output.

If the **−V** option is supplied, **compgen** stores the generated completions into the indexed array variable *varname* instead of writing them 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 or stored.

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

complete [**−abcdefgjksuv**] [**−o** *comp-option*] [**−DEI**] [**−A** *action*]
[**−G** *globpat*] [**−W** *wordlist*] [**−F** *function*] [**−C** *command*]
[**−X** *filterpat*] [**−P** *prefix*] [**−S** *suffix*] *name* [*name* ...]
complete −pr [**−DEI**] [*name* ...]
Specify how arguments to each *name* should be completed.

If the **−p** option is supplied, or if no options or *name*s are supplied, print existing completion specifications 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 other supplied options and actions should apply to the command completion; that is, completion attempted on a command for which no completion has previously been defined. The **−E** option indicates that other supplied options and actions should apply to command completion; that is, completion attempted on a blank line. The **−I** option indicates that other supplied options and actions should apply to completion on the initial non-assignment word on the line, or after a command delimiter such as **;** or **|**, which is usually command name completion. If multiple options are supplied, the **−D** option takes precedence over **−E**, and both take precedence over **−I**. If any of **−D**, **−E**, or **−I** are supplied, any other *name* arguments are ignored; these completions only apply to the case specified by the option.

The process of applying these completion specifications when attempting word completion is described in *bash*(1).

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 filename−specific processing (such as adding a slash to directory names, quoting special characters, or suppressing trailing spaces). This is intended to be used with shell functions.

fullquote
Tell **readline** to quote all the completed words even if they are not filenames.

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

nosort Tell **readline** not to sort the list of possible completions alphabetically.

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

plusdirs After generating any matches defined by the compspec, attempt directory name completion and add any matches 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. Arguments are passed as with the **−F** option.

−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 *function* finishes, programmable completion retrieves the possible completions from the value of the **COMPREPLY** array variable.

−G *globpat*
Expand the pathname expansion pattern *globpat* to generate the possible completions.

−P *prefix*
Add *prefix* to the beginning of each possible completion after all other options have been applied.

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

−W *wordlist*
Split the *wordlist* using the characters in the **IFS** special variable as delimiters, and expand each resulting word. Shell quoting is honored within *wordlist*, in order to provide a mechanism for the words to contain shell metacharacters or characters in the value of **IFS**. The possible completions are the members of the resultant list which match a prefix of 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**, **−r**, **−D**, **−E**, or **−I** 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*] [**−DEI**] [**+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 supplied, 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 other supplied options should apply to the command completion; the **−E** option indicates that other supplied options should apply to command completion; and the **−I** option indicates that other supplied options should apply to completion on the initial word on the line. These are determined in the same way as the **complete** builtin.

If multiple options are supplied, the **−D** option takes precedence over **−E**, and both take precedence over **−I**.

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*]
continue resumes the next iteration of the enclosing **for**, **while**, **until**, or **select** loop. If *n* is specified, **bash** resumes the *n*th enclosing loop. *n* must be ≥ 1. If *n* is greater than the number of enclosing loops, the shell resumes the last enclosing loop (the loop). The return value is 0 unless *n* is not greater than or equal to 1.

declare [**−aAfFgiIlnrtux**] [**−p**] [*name*[=*value*] ...]
typeset [**−aAfFgiIlnrtux**] [**−p**] [*name*[=*value*] ...]

Declare variables and/or give them attributes. If no *name*s are given then display the values of variables or functions. The **−p** option will display the attributes and values of each *name*. When **−p** is used with *name* arguments, additional options, other than **−f** and **−F**, are ignored.

When **−p** is supplied without *name* arguments, **declare** 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 restricts 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 each *name* 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 when **declare** is not executed in a shell function.

The **−I** option causes local variables to inherit the attributes (except the *nameref* attribute) and value of any existing variable with the same *name* at a surrounding scope. If there is no existing variable, the local variable is initially unset.

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** in *bash*(1)).
−A Each *name* is an associative array variable (see **Arrays** in *bash*(1)).
−f Each *name* refers to a shell function.
−i The variable is treated as an integer; arithmetic evaluation (see **ARITHMETIC EVALUATION** in *bash*(1)) 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, assignments, and attribute modifications to *name*, except those using or changing the **−n** attribute itself, are performed on the variable referenced by *name*'s value. The nameref 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 each *name* for export to subsequent commands via the environment.

Using instead of turns off the specified attribute instead, with the exceptions that **+a** and **+A** may not be used to destroy array variables 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*. When using **−a** or **−A** and the compound assignment syntax to create array variables, additional attributes do not take effect until subsequent assignments.

The return value is 0 unless an invalid option is encountered, an attempt is made to define a function using 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** in *bash*(1)), 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, display 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. The current directory is always the first directory in the stack.

Options, if supplied, have the following meanings:

−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**] [*id* . . .]

Without options, remove each *id* from the table of active jobs. Each *id* may be a job specification *jobspec* or a process ID *pid*; if *id* is a *pid*, **disown** uses the job containing *pid* as *jobspec*.

If the **−h** option is supplied, **disown** does not remove the jobs corresponding to each *id* from the jobs table, but rather marks them so the shell does not send **SIGHUP** to the job if the shell receives a **SIGHUP**.

If no *id* is supplied, the **−a** option means to remove or mark all jobs; the **−r** option without an *id* argument removes or marks running jobs. If no *id* is supplied, and neither the **−a** nor the **−r** option is supplied, **disown** removes or marks the current job.

The return value is 0 unless an *id* 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 not printed.

If the **−e** option is given, **echo** interprets the following backslash-escaped characters. The **−E** option disables interpretation of these escape characters, even on systems where they are interpreted by default. The **xpg_echo** shell option determines whether or not **echo** interprets any options and expands these escape characters. **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).

echo writes any unrecognized backslash-escaped characters unchanged.

enable [**−a**] [**−dnps**] [**−f** *filename*] [*name* . . .]
Enable and disable builtin shell commands. Disabling a builtin allows an executable file 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 files.

If **−n** is supplied, each *name* is disabled; otherwise, *name*s are enabled. For example, to use the **test** binary found usin g **PATH** instead of the shell builtin version, run

If no *name* arguments are supplied, or if the **−p** option is supplied, print a list of shell builtins. With no other option arguments, the list consists of all enabled shell builtins. If **−n** is supplied, print only disabled builtins. If **−a** is supplied, the list printed includes all builtins, with an indication of whether or not each is enabled. The **−s** option means to restrict the output to the POSIX *special* builtins.

The **−f** option means to load the new builtin command *name* from shared object *filename*, on systems that support dynamic loading. If *filename* does not contain a slash, **Bash** will use the value of the **BASH_LOADABLES_PATH** variable as a colon-separated list of directories in which to search for *filename*. The default for **BASH_LOADABLES_PATH** is system-dependent, and may include to force a search of the current directory. The **−d** option will delete a builtin previously loaded with **−f**. If *−s* is used with *−f*, the new builtin becomes a POSIX special builtin.

If no options are supplied and a *name* is not a shell builtin, **enable** will attempt to load *name* from a shared object named *name*, as if the command were

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* . . .]
Concatenate the *arg*s together into a single command, separating them with spaces. **Bash** then reads and execute this command, and returns its exit status as the return status 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 without creating a new process. *command* cannot be a shell builtin or function. 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 a non-zero status. An interactive shell returns a non-zero status if the file cannot be executed. A subshell exits unconditionally if **exec** fails.

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. Any trap on **EXIT** is executed before the shell terminates.

export [**−fn**] [*name*[=*value*]] . . .
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.

The **−n** option unexports, or removes the export attribute, from each *name*. If no *names* are given, or if the **−p** option is supplied, **export** prints a list of names of all exported variables on the standard output.

export allows the value of a variable to be set when it is exported or unexported by following the variable name with =*value*. This sets the value of the variable to *value* while modifying the export attribute. **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.

false Does nothing; returns a non-zero status.

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).

When listing, a *first* or *last* of 0 is equivalent to −1 and −0 is equivalent to the current command (usually the **fc** command); otherwise 0 is equivalent to −1 and −0 is invalid. If *last* is not specified, it is set to the current command for listing (so that 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.

If the **−l** option is supplied, the commands are listed on the standard output. The **−n** option suppresses the command numbers when listing. The **−r** option reverses the order of the commands.

Otherwise, **fc** invokes the editor named by *ename* on a file containing those commands. If *ename* is not supplied, **fc** uses the value of the **FCEDIT** variable, and the value of **EDITOR** if **FCEDIT** is not set. If neither variable is set, **fc** uses *vi*. When editing is complete, **fc** reads the file containing the edited commands and echoes and executes them.

In the second form, **fc** re-executes *command* after replacing each instance of *pat* with *rep*. *Command* is interpreted the same as *first* above.

A useful alias to use with **fc** is so that typing runs the last command beginning with and typing re-executes the last command.

If the first form is used, the return value is zero unless an invalid option is encountered or *first* or *last* specify history lines out of range. When editing and re-executing a file of commands, the return value is the value of the last command executed or failure if an error occurs with the temporary file. If the second form is used, the return status is that of the re-executed command, unless *cmd* does not specify a valid history entry, in which case **fc** returns a non-zero status.

fg [*jobspec*]

Resume *jobspec* in the foreground, and make it the current job. If *jobspec* is not present, **fg** uses the shell's notion of the *current job*. 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* [*arg* ...]

getopts is used by shell scripts and functions to parse positional parameters and obtain options and their arguments. *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 to use a new set of parameters.

When it reaches the end of options, **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 supplied as *arg* values, **getopts** parses those instead.

getopts can report errors in two ways. If the first character of *optstring* is a colon, **getopts** uses *silent* error reporting. In normal operation, **getopts** prints diagnostic messages when it encounters invalid options or missing option arguments. 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 **getopts** detects an invalid option, it places ? into *name* and, if not silent, prints an error message and unsets **OPTARG**. If **getopts** is silent, it assigns the option character found to **OPTARG** and does not print a diagnostic message.

If a required argument is not found, and **getopts** is not silent, it sets the value of *name* to a question mark (**?**), unsets **OPTARG**, and prints a diagnostic message. If **getopts** is silent, it sets the value of *name* to a colon (**:**) and sets **OPTARG** 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, it remembers the full pathname of the command *name* as determined by searching the directories in **$PATH**. Any previously-remembered pathname associated with *name* is discarded. If the **−p** option is supplied, **hash** uses *filename* as the full pathname of the command.

The **−r** option causes the shell to forget all remembered locations. Assigning to the **PATH** variable also clears all hashed filenames. The **−d** option causes the shell to forget the remembered location of each *name*.

If the **−t** option is supplied, **hash** prints the full pathname corresponding to each *name*. If multiple *name* arguments are supplied with **−t**, **hash** prints the *name* before the corresponding hashed full pathname. The **−l** option displays output in a format that may be reused as input.

If no arguments are given, or if only **−l** is supplied, **hash** prints information about remembered commands. The **−t**, **−d**, and **−p** options (the options that act on the *name* arguments) are mutually exclusive. Only one will be active. If more than one is supplied, **−t** has higher priority than **−p**, and both have higher priority than **−d**.

The return status is zero 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 it displays a list of all the builtins and shell compound commands.

−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 −d *start-end*
history −anrw [*filename*]

history −p *arg* [*arg* ...]
history −s *arg* [*arg* ...]

With no options, display the command history list with numbers. Entries prefixed with a ***** have been modified. An argument of *n* lists only the last *n* entries. 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. If **history** uses **HISTTIMEFORMAT**, it does not print an intervening space between the formatted time stamp and the history entry.

If *filename* is supplied, **history** uses it as the name of the history file; if not, it uses the value of **HISTFILE**. If *filename* is not supplied and **HISTFILE** is unset or null, the **−a, −n, −r,** and **−w** options have no effect.

Options, if supplied, have the following meanings:

−c Clear the history list by deleting all the entries. This can be used with the other options to replace the history list.

−d *offset*
Delete the history entry at position *offset*. If *offset* is negative, it is interpreted as relative to one greater than the last history position, so negative indices count back from the end of the history, and an index of −1 refers to the current **history −d** command.

−d *start−end*
Delete the range of history entries between positions *start* and *end*, inclusive. Positive and negative values for *start* and *end* are interpreted as described above.

−a Append the history lines to the history file. These are history lines entered since the beginning of the current **bash** session, but not already appended to the history file.

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

−r Read the history file and append its contents to the current history list.

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

−p Perform history substitution on the following *args* and display the result on the standard output, without storing 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 adding the *args*.

If the **HISTTIMEFORMAT** variable is set, **history** writes the time stamp information associated with each history entry to the history file, marked with the history comment character as described above. When the history file is read, lines beginning with the history comment character followed immediately by a digit are interpreted as timestamps for the following history entry.

The return value is 0 unless an invalid option is encountered, an error occurs while reading or writing the history file, an invalid *offset* or range 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 supplied, **jobs** restricts output 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*] *id* [. . .]
kill −l|−L [*sigspec* | *exit_status*]
Send the signal specified by *sigspec* or *signum* to the processes named by each *id*. Each *id* may be a job specification *jobspec* or a process ID *pid*. *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 supplied, then **kill** sends **SIGTERM**.

The **−l** option lists the signal names. If any arguments are supplied when **−l** is given, **kill** lists the names of the signals corresponding to the arguments, 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; if it is supplied, **kill** prints the name of the signal that caused the process to terminate. **kill** assumes that process exit statuses are greater than 128; anything less than that is a signal number. The **−L** option is equivalent to **−l**.

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 evaluated as an arithmetic expression (see **ARITHMETIC EVALUATION** in *bash*(1)). If the last *arg* evaluates to 0, **let** returns 1; otherwise **let** returns 0.

local [*option*] [*name*[=*value*] . . . | −]
For each argument, create a local variable named *name* and assign it *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. It is an error to use **local** when not within a function.

If *name* is −, it makes the set of shell options local to the function in which **local** is invoked: any shell options changed using the **set** builtin inside the function after the call to **local** are restored to their original values when the function returns. The restore is performed as if a series of **set** commands were executed to restore the values that were in place before the function.

With no operands, **local** writes a list of local variables to the standard output.

The return status is 0 unless **local** is used outside a function, an invalid *name* is supplied, or *name* is a readonly variable.

logout [*n***]**
Exit a login shell, returning a status of *n* to the shell's parent.

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

- **−d** Use the first character of *delim* to terminate each input line, rather than newline. If *delim* is the empty string, **mapfile** will terminate a line when it reads a NUL character.
- **−n** Copy at most *count* lines. If *count* is 0, copy all lines.
- **−O** Begin assigning to *array* at index *origin*. The default index is 0.
- **−s** Discard the first *count* lines read.
- **−t** Remove a trailing *delim* (default 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 zero 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*]

Remove entries from the directory stack. The elements are numbered from 0 starting at the first directory listed by **dirs**, so **popd** is equivalent to With no arguments, **popd** removes the top directory from the stack, and changes to the new top directory. Arguments, if supplied, have the following meanings:

−n Suppress the normal change of directory when removing directories from the stack, only manipulate the stack.

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

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

If the top element of the directory stack is modified, and the −*n* option was not supplied, **popd** uses the **cd** builtin to change to the directory at the top of the stack. If the **cd** fails, **popd** returns a non-zero value.

Otherwise, **popd** returns false if an invalid option is supplied, the directory stack is empty, or *n* specifies a non-existent directory stack entry.

If the **popd** command is successful, **bash** runs **dirs** to show the final contents of the directory stack, and the return status is 0.

printf [**−v** *var*] *format* [*arguments*]

Write the formatted *arguments* to the standard output under the control of the *format*. The **−v** option assigns the output to the variable *var* rather than printing it 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*(3) format characters **cCsSndiouxXeEfFgGaA**, **printf** interprets the following additional format specifiers:

%b causes **printf** to expand backslash escape sequences in the corresponding *argument* in the same way as **echo −e**.

%q causes **printf** to output the corresponding *argument* in a format that can be reused as shell input. **%q** and **%Q** use the **$** quoting style if any characters in the argument string require it, and backslash quoting otherwise. If the format string uses the *printf* alternate form, these two formats quote the argument string using single quotes.

%Q like **%q**, but applies any supplied precision to the *argument* before quoting it.

%(*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. This format specifier recognizes two special argument values: −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 supplied. This is an exception to the usual **printf** behavior.

The %b, %q, and %T format specifiers all use the field width and precision arguments from the format specification and write that many bytes from (or use that wide a field for) the expanded argument, which usually contains more characters than the original.

The %n format specifier accepts a corresponding argument that is treated as a shell variable name.

The %s and %c format specifiers accept an l (long) modifier, which forces them to convert the argument string to a wide-character string and apply any supplied field width and precision in terms of characters, not bytes. The %S and %C format specifiers are equivalent to %ls and %lc,

respectively.

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 numeric value of the following character, using the current locale.

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 if an invalid option is supplied or a write or assignment error occurs.

pushd [**−n**] [+*n*] [−*n*]
pushd [**−n**] [*dir*]

Add a directory to the top of the directory stack, or rotate the stack, making the new top of the stack the current working directory. With no arguments, **pushd** exchanges the top two elements of the directory stack. Arguments, if supplied, have the following meanings:

−n Suppress the normal change of directory when rotating or adding directories to the stack, only manipulate the stack.

+*n* Rotate 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.

After the stack has been modified, if the **−n** option was not supplied, **pushd** uses the **cd** builtin to change to the directory at the top of the stack. If the **cd** fails, **pushd** returns a non-zero value.

Otherwise, if no arguments are supplied, **pushd** returns zero unless the directory stack is empty. When rotating the directory stack, **pushd** returns zero unless the directory stack is empty or *n* specifies a non-existent directory stack element.

If the **pushd** command is successful, **bash** runs **dirs** to show the final contents of the directory stack.

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 [**−Eers**] [**−a** *aname*] [**−d** *delim*] [**−i** *text*] [**−n** *nchars*] [**−N** *nchars*] [**−p** *prompt*] [**−t** *timeout*] [**−u** *fd*] [*name* ...]

Read one line from the standard input, or from the file descriptor *fd* supplied as an argument to the **−u** option, split it into words as described in *bash* (1) under **Word Splitting**, and assign the first word to the first *name*, the second word to the second *name*, and so on. If there are more words than names, the remaining words and their intervening delimiters are 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 the value of the **IFS** variable are used to split the line into words using the same rules the shell uses for expansion (described in *bash* (1) under **Word Splitting**). The backslash character (\) removes any special meaning for the next character read and is used 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* terminates the input line, rather than newline. If *delim* is the empty string, **read** will terminate a line when it reads a NUL character.

−e If the standard input is coming from a terminal, **read** uses **readline** (see **READLINE** in *bash*(1)) to obtain the line. **Readline** uses the current (or default, if line editing was not previously active) editing settings, but uses **readline**'s default filename completion.

−E If the standard input is coming from a terminal, **read** uses **readline** (see **READLINE** in *bash*(1)) to obtain the line. **Readline** uses the current (or default, if line editing was not previously active) editing settings, but uses bash's default completion, including programmable completion.

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

−n *nchars*
read returns after reading *nchars* characters rather than waiting for a complete line of input, unless it encounters EOF or **read** times out, but honors a delimiter if it reads fewer than *nchars* characters before the delimiter.

−N *nchars*
read returns after reading exactly *nchars* characters rather than waiting for a complete line of input, unless it encounters EOF or **read** times out. Any delimiter characters in the input are not treated specially and do not cause **read** to return until it has read *nchars* characters. The result is not split on the characters in **IFS**; the intent is that the variable is assigned exactly the characters read (with the exception of backslash; see the **−r** option below).

−p *prompt*
Display *prompt* on standard error, without a trailing newline, before attempting to read any input, but 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 then 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 it does not read a complete line of input (or a specified number of characters) 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, it saves any partial input read into the specified variable *name*, and the exit status is greater than 128. If *timeout* is 0, **read** returns immediately, without trying to read any data. In this case, the exit status is 0 if input is available on the specified file descriptor, or the read will return EOF, non-zero otherwise.

−u *fd* Read input from file descriptor *fd* instead of the standard input.

Other than the case where *delim* is the empty string, **read** ignores any NUL characters in the input.

If no *names* are supplied, **read** assigns the line read, without the ending delimiter but otherwise unmodified, to the variable **REPLY**.

The exit status is zero, unless end-of-file is encountered, **read** times out (in which case the status 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 or unset. If the **−f** option is supplied, each *name* refers to a shell function. 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 supplied, or if the **−p** option is supplied, print a list of all readonly names. The other options may be used to restrict the output to a subset of the set of readonly names. The **−p** option displays output in a format that may be reused as input.

readonly allows the value of a variable to be set at the same time the readonly attribute is changed by following the variable name with =*value*. This sets the value of the variable is to *value* while modifying the readonly attribute.

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*]

Stop executing a shell function or sourced file and return the value specified by *n* to its caller. If *n* is omitted, the return status is that of the last command executed. If **return** is executed by a trap handler, the last command used to determine the status is the last command executed before the trap handler. If **return** is executed during a **DEBUG** trap, the last command used to determine the status is the last command executed by the trap handler before **return** was invoked.

When **return** is used to terminate execution of a script being executed 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.

Any command associated with the **RETURN** trap is executed before execution resumes after the function or script.

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**.

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

Without options, display the name and value of each shell variable 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 Each variable or function that is created or modified is given the export attribute and marked for export to the environment of subsequent commands.

−b Report the status of terminated background jobs immediately, rather than before the next primary prompt or after a foreground command terminates. 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** in *bash*(1)), 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 (subject to the state of the **pipefail** shell option), 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** in *bash*(1)), 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** in *bash*(1)). 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 in *bash*(1) under **HISTORY**. This option is on by default in interactive shells.

ignoreeof
The effect is as if the shell command had been executed (see **Shell Variables** in *bash*(1)).

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*). See **SEE ALSO** in *bash*(1) for a reference to a document that details how posix mode affects bash's behavior.

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*, **set** prints the current shell option settings. If **+o** is supplied with no *option−name*, **set** prints a series of **set** commands to recreate the current option settings on the standard output.

−p Turn on *privileged* mode. In this mode, the shell does not read the **$ENV** and **$BASH_ENV** files, 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.

−r Enable restricted shell mode. This option cannot be unset once it has been set.

−t Exit after reading and executing one command.

−u Treat unset variables and parameters other than the special parameters and or array variables subscripted with or 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, to the standard error.

−B The shell performs brace expansion (see **Brace Expansion** in *bash*(1)). This is on by default.

−C If set, **bash** does not overwrite an existing file with the **>**, **>&**, and **<>** redirection operators. Using the redirection operator **>|** instead of **>** will override this and force the creation of an output file.

−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, unset the positional parameters. Otherwise, set the positional parameters to the *arg*s, even if some of them begin with a **−**.

− Signal the end of options, and assign all remaining *arg*s 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 zero unless an invalid option is encountered.

shift [*n*]

Rename positional parameters from *n*+1 ... 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 settings controlling optional shell behavior. The settings can be either those listed below, or, if the **−o** option is used, those available with the **−o** option to the **set** builtin command.

With no options, or with the **−p** option, display a list of all settable options, with an indication of whether or not each is set; if any *optnames* are supplied, the output is restricted to those options. The **−p** option displays output 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 supplied 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:

array_expand_once
If set, the shell suppresses multiple evaluation of associative and indexed array subscripts during arithmetic expression evaluation, while executing builtins that can perform variable assignments, and while executing builtins that perform array dereferencing.

assoc_expand_once
Deprecated; a synonym for **array_expand_once**.

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.

bash_source_fullpath
If set, filenames added to the **BASH_SOURCE** array variable are converted to full pathnames (see **Shell Variables** above).

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, the **cd** command attempts to correct minor errors in the spelling of a directory component. Minor errors include transposed characters, a missing character, and one extra character. If **cd** corrects the directory name, it prints the corrected filename, 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, **bash** performs a normal path search.

checkjobs
If set, **bash** lists the status of any stopped and running jobs before exiting an interactive shell. If any jobs are running, **bash** defers the exit until a second exit is attempted without an intervening command (see **JOB CONTROL** in *bash*(1)). The shell always postpones exiting if any jobs are stopped.

checkwinsize
If set, **bash** checks the window size after each external (non-builtin) command and, if necessary, updates the values of **LINES** and **COLUMNS**, using the file descriptor associated with the standard error if it is a terminal. This option is enabled by default.

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. This option is enabled by default, but only has an effect if command history is enabled, as described in *bash*(1) under **HISTORY**.

compat31
compat32
compat40
compat41
compat42
compat43
compat44
compat50
These control aspects of the shell's compatibility mode (see **SHELL COMPATIBILITY MODE** in *bash*(1)).

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. The filenames . and .. must always be matched explicitly, even if **dotglob** is set.

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

expand_aliases
If set, aliases are expanded as described in *bash*(1) under **ALIASES**. This option is enabled by default for interactive shells.

extdebug
If set at shell invocation, or in a shell startup file, arrange to execute the debugger profile before the shell starts, identical to the **−−debugger** option. If set after invocation, 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), the shell simulates a call to **return**.
4. **BASH_ARGC** and **BASH_ARGV** are updated as described in their descriptions in *bash*(1)).

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, enable the extended pattern matching features described in *bash*(1) under **Pathname Expansion**.

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** in *bash*(1) for a description of **FIGNORE**. This option is enabled by default.

globasciiranges
If set, range expressions used in pattern matching bracket expressions (see **Pattern Matching** in *bash*(1)) behave as if in the traditional C locale when performing comparisons. That is, pattern matching does not take the current locale's collating sequence into account, so **b** will not collate between **A** and **B**, and upper-case and lower-case ASCII characters will collate together.

globskipdots
If set, pathname expansion will never match the filenames . and .. even if the pattern begins with a This option is enabled by default.

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, the 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** in *bash*(1)). This is enabled by default.

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

inherit_errexit
If set, command substitution inherits the value of the **errexit** option, instead of unsetting it in the subshell environment. This option is enabled when posix mode is enabled.

interactive_comments
In an interactive shell, a word beginning with **#** causes that word and all remaining characters on that line to be ignored, as in a non-interactive shell (see **COMMENTS** in *bash*(1)). 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.

localvar_inherit
If set, local variables inherit the value and attributes of a variable of the same name that exists at a previous scope before any new value is assigned. The nameref attribute is not inherited.

localvar_unset
If set, calling **unset** on local variables in previous function scopes marks them so subsequent lookups find them unset until that function returns. This is identical to the behavior of unsetting local variables at the current function scope.

login_shell
The shell sets this option if it is started as a login shell (see **INVOCATION** in *bash*(1)). 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, **bash** displays the message

no_empty_cmd_completion
If set, and **readline** is being used, **bash** does not search **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** in *bash*(1)).

nocasematch
If set, **bash** matches patterns in a case−insensitive fashion when performing matching while executing **case** or **[[** conditional commands, when performing pattern substitution word expansions, or when filtering possible completions as part of programmable completion.

noexpand_translation
If set, **bash** encloses the translated results of **$**... quoting in single quotes instead of double quotes. If the string is not translated, this has no effect.

nullglob
If set, pathname expansion patterns which match no files (see **Pathname Expansion** in *bash*(1)) expand to nothing and are removed, rather than expanding to themselves.

patsub_replacement
If set, **bash** expands occurrences of **&** in the replacement string of pattern substitution to the text matched by the pattern, as described under **Parameter Expansion** in *bash*(1). This option is enabled by default.

progcomp
If set, enable the programmable completion facilities (see **Programmable Completion** in *bash*(1)). This option is enabled by default.

progcomp_alias
If set, and programmable completion is enabled, **bash** treats a command name that doesn't have any completions as a possible alias and attempts alias expansion. If it has an alias, **bash** attempts programmable completion using the command word resulting from the expanded alias.

promptvars
If set, prompt strings undergo parameter expansion, command substitution, arithmetic expansion, and quote removal after being expanded as described in **PROMPTING** in *bash*(1). This option is enabled by default.

restricted_shell
The shell sets this option if it is started in restricted mode (see **RESTRICTED SHELL** in *bash*(1)). 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 when the **−p** option is not supplied. This option is enabled by default.

varredir_close
If set, the shell automatically closes file descriptors assigned using the *{varname}* redirection syntax (see **REDIRECTION** in *bash*(1)) instead of leaving them open when the command completes.

xpg_echo
If set, the **echo** builtin expands backslash-escape sequences by default. If the **posix** shell option is also enabled, **echo** does not interpret any options.

suspend [**−f**]
Suspend the execution of this shell until it receives a **SIGCONT** signal. A login shell, or a shell without job control enabled, cannot be suspended; the **−f** option will override this and force the suspension. The return status is 0 unless the shell is a login shell or job control is not enabled and **−f** is not supplied.

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 in *bash*(1) 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. **test** uses operator precedence 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 normal operator precedence.
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 in *bash*(1) 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 in *bash*(1) 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
: The following conditions are applied in the order listed. If the first argument is **!**, the result is the negation of the three-argument expression composed of the remaining arguments. If the first argument is exactly **(** and the fourth argument is exactly **)**, the result is the two-argument test of the second and third 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 the shell is in posix mode, or if the expression is part of the **[[** command, the **<** and **>** operators sort using the current locale. If the shell is not in posix mode, the **test** and **[** commands sort lexicographically using ASCII ordering.

The historical operator-precedence parsing with 4 or more arguments can lead to ambiguities when it encounters strings that look like primaries. The POSIX standard has deprecated the **−a** and **−o** primaries and enclosing expressions within parentheses. Scripts should no longer use them. It's much more reliable to restrict test invocations to a single primary, and to replace uses of **−a** and **−o** with the shell's **&&** and **||** list operators.

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

trap [**−lpP**] [[*action*] *sigspec* . . .]
: The *action* is a command that is read and executed when the shell receives any of the signals *sigspec*. If *action* is absent (and there is a single *sigspec*) or **−**, each specified *sigspec* is reset to the value it had when the shell was started. If *action* is the null string the signal specified by each *sigspec* is ignored by the shell and by the commands it invokes.

If no arguments are supplied, **trap** displays the actions associated with each trapped signal as a set of **trap** commands that can be reused as shell input to restore the current signal dispositions. If **−p** is given, and *action* is not present, then **trap** displays the actions associated with each *sigspec* or, if none are supplied, for all trapped signals, as a set of **trap** commands that can be reused as shell input to restore the current signal dispositions. The **−P** option behaves similarly, but displays only the actions associated with each *sigspec* argument. **−P** requires at least one *sigspec* argument. The **−P** or **−p** options may be used in a subshell environment (e.g., command substitution) and, as long as they are used before **trap** is used to change a signal's handling, will display the state of its parent's traps.

The **−l** option prints 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 **−l** is supplied with no *sigspec* arguments, it prints a list of valid signal names.

If a *sigspec* is **EXIT** (0), *action* is executed on exit from the shell. If a *sigspec* is **DEBUG**, *action* is executed before every *simple command*, *for* command, *case* command, *select* command, ((arithmetic command, [[conditional command, arithmetic *for* command, and before the first command executes in a shell function (see **SHELL GRAMMAR** in *bash*(1)). Refer to the description of the **extdebug** shell option (see **shopt** in *bash*(1)) for details of its effect on the **DEBUG** trap. If a *sigspec* is **RETURN**, *action* is executed each time a shell function or a script executed with the **.** or **source** builtins finishes executing.

If a *sigspec* is **ERR**, *action* is executed whenever 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 (subject to the state of the **pipefail** shell option), or if the command's return

value is being inverted using **!**. These are the same conditions obeyed by the **errexit** (**−e**) option.

When the shell is not interactive, signals ignored upon entry to the shell cannot be trapped or reset. Interactive shells permit trapping signals ignored on entry. 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.

true Does nothing, returns a 0 status.

type [**−aftpP**] *name* [*name* ...]
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 executable file, respectively. If the *name* is not found, **type** prints nothing and returns a non-zero exit status.

If the **−p** option is used, **type** either returns the pathname of the executable file that would be found by searching **$PATH** for *name* or nothing if would not return *file*. The **−P** option forces a **PATH** search for each *name*, even if would not return *file*. If *name* is present in the table of hashed commands, **−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 a command named *name*. This includes aliases, reserved words, functions, and builtins, but the path search options (**−p** and **−P**) can be supplied to restrict the output to executable files. **type** does not consult the table of hashed commands when using **−a** with **−p**, and only performs a **PATH** search for *name*.

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 [**−HS**] **−a**
ulimit [**−HS**] [**−bcdefiklmnpqrstuvxPRT** [*limit*]]
Provides control over the resources available to the shell and to processes it starts, on systems that allow such control.

The **−H** and **−S** options specify whether 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, **ulimit** sets both the soft and hard limits.

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, **ulimit** prints the current value of the soft limit of the resource, unless the **−H** option is given. When more than one resource is specified, the limit name and unit, if appropriate, are printed before the value. Other options are interpreted as follows:

−a Report all current limits; no limits are set.
−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 (
−f The maximum size of files written by the shell and its children.
−i The maximum number of pending signals.
−k The maximum number of kqueues that may be allocated.
−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.
−P The maximum number of pseudoterminals.
−R The maximum time a real-time process can run before blocking, in microseconds.
−T The maximum number of threads.

If *limit* is supplied, and the **−a** option is not used, *limit* is the new value of the specified resource. If no option is supplied, then **−f** is assumed.

Values are in 1024-byte increments, except for **−t**, which is in seconds; **−R**, which is in microseconds; **−p**, which is in units of 512-byte blocks; **−P**, **−T**, **−b**, **−k**, **−n**, and **−u**, which are unscaled values; and, when in posix mode, **−c** and **−f**, which are in 512-byte increments. 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*]
Set the user file-creation mask 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, **umask** prints the current value of the mask. The **−S** option without a *mode* argument prints the mask in a symbolic format; 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 zero if the mode was successfully changed or if no *mode* argument was supplied, and non-zero otherwise.

unalias [**−a**] [*name* . . .]
Remove each *name* from the list of defined aliases. If **−a** is supplied, remove all alias definitions. 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. 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. Read-only variables and functions may not be unset. When variables or functions are removed, they are also removed from the environment passed to subsequent commands. If no options are supplied, each *name* refers to a variable; if there is no variable by that name, a function with that name, if any, is unset. Some shell variables may not be unset. If any of **BASH_ALIASES**, **BASH_ARGV0**, **BASH_CMDS**, **BASH_COMMAND**, **BASH_SUBSHELL**, **BASHPID**, **COMP_WORDBREAKS**, **DIRSTACK**, **EPOCHREALTIME**, **EPOCHSECONDS**, **FUNCNAME**, **GROUPS**, **HISTCMD**, **LINENO**, **RANDOM**, **SECONDS**, or **SRANDOM** are unset, they lose their special properties, even if they are subsequently reset. The exit status is true unless a *name* is readonly or may not be unset.

wait [**−fn**] [**−p** *varname*] [*id* . . .]
Wait for each specified child process *id* and return the termination status of the last *id*. Each *id* may be a process ID *pid* or a job specification *jobspec*; if a jobspec is supplied, **wait** waits for all processes in the job.

If no options or *id*s are supplied, **wait** waits for all running background jobs and the last-executed process substitution, if its process id is the same as **$!**, and the return status is zero.

If the **−n** option is supplied, **wait** waits for any one of the given *id*s or, if no *id*s are supplied, any job or process substitution, to complete and returns its exit status. If none of the supplied *id*s is a child of the shell, or if no *id*s are supplied and the shell has no unwaited-for children, the exit status is 127.

If the **−p** option is supplied, **wait** assigns the process or job identifier of the job for which the exit status is returned to the variable *varname* named by the option argument. The variable, which cannot be readonly, will be unset initially, before any assignment. This is useful only when used with the **−n** option.

Supplying the **−f** option, when job control is enabled, forces **wait** to wait for each *id* to terminate before returning its status, instead of returning when it changes status.

If none of the *id*s specify one of the shell's active child processes, the return status is 127. If **wait** is interrupted by a signal, any *varname* will remain unset, and the return status will be greater than 128, as described under **SIGNALS** in *bash*(1). Otherwise, the return status is the exit status of the last *id*.

SHELL COMPATIBILITY MODE

Bash-4.0 introduced the concept of a *shell compatibility level*, specified as a set of options to the shopt builtin (**compat31**, **compat32**, **compat40**, **compat41**, and so on). There is only one current compatibility level — each option is mutually exclusive. The compatibility level is intended to allow users to select behavior from previous versions that is incompatible with newer versions while they migrate scripts to use current features and behavior. It's intended to be a temporary solution.

This section does not mention behavior that is standard for a particular version (e.g., setting **compat32** means that quoting the right hand side of the regexp matching operator quotes special regexp characters in the word, which is default behavior in bash-3.2 and subsequent versions).

If a user enables, say, **compat32**, it may affect the behavior of other compatibility levels up to and including the current compatibility level. The idea is that each compatibility level controls behavior that changed in that version of **bash**, but that behavior may have been present in earlier versions. For instance, the change to use locale-based comparisons with the **[[** command came in bash-4.1, and earlier versions used ASCII-based comparisons, so enabling **compat32** will enable ASCII-based comparisons as well. That granularity may not be sufficient for all uses, and as a result users should employ compatibility levels carefully. Read the documentation for a particular feature to find out the current behavior.

Bash-4.3 introduced a new shell variable: **BASH_COMPAT**. The value assigned to this variable (a decimal version number like 4.2, or an integer corresponding to the **compat***NN* option, like 42) determines the compatibility level.

Starting with bash-4.4, **bash** began deprecating older compatibility levels. Eventually, the options will be removed in favor of **BASH_COMPAT**.

Bash-5.0 was the final version for which there was an individual shopt option for the previous version. **BASH_COMPAT** is the only mechanism to control the compatibility level in versions newer than bash-5.0.

The following table describes the behavior changes controlled by each compatibility level setting. The **compat***NN* tag is used as shorthand for setting the compatibility level to *NN* using one of the following mechanisms. For versions prior to bash-5.0, the compatibility level may be set using the corresponding **compat***NN* shopt option. For bash-4.3 and later versions, the **BASH_COMPAT** variable is preferred, and it is required for bash-5.1 and later versions.

compat31

- Quoting the rhs of the **[[** command's regexp matching operator (=) has no special effect.

compat32

- The **<** and **>** operators to the **[[** command do not consider the current locale when comparing strings; they use ASCII ordering.

compat40

- The **<** and **>** operators to the **[[** command do not consider the current locale when comparing strings; they use ASCII ordering. **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).

compat41

- In *posix* mode, **time** may be followed by options and still be recognized as a reserved word (this is POSIX interpretation 267).
- In *posix* mode, the parser requires that an even number of single quotes occur in the *word* portion of a double-quoted parameter expansion and treats them specially, so that characters within the single quotes are considered quoted (this is POSIX interpretation 221).

compat42

- The replacement string in double-quoted pattern substitution does not undergo quote removal, as it does in versions after bash-4.2.
- In posix mode, single quotes are considered special when expanding the *word* portion of a double-quoted parameter expansion and can be used to quote a closing brace or other special character (this is part of POSIX interpretation 221); in later versions, single quotes are not special within double-quoted word expansions.

compat43

- Word expansion errors are considered non-fatal errors that cause the current command to fail, even in posix mode (the default behavior is to make them fatal errors that cause the shell to exit).
- When executing a shell function, the loop state (while/until/etc.) is not reset, so **break** or **continue** in that function will break or continue loops in the calling context. Bash-4.4 and later reset the loop state to prevent this.

compat44

- The shell sets up the values used by **BASH_ARGV** and **BASH_ARGC** so they can expand to the shell's positional parameters even if extended debugging mode is not enabled.
- A subshell inherits loops from its parent context, so **break** or **continue** will cause the subshell to exit. Bash-5.0 and later reset the loop state to prevent the exit
- Variable assignments preceding builtins like **export** and **readonly** that set attributes continue to affect variables with the same name in the calling environment even if the shell is not in posix mode.

compat50

- Bash-5.1 changed the way **$RANDOM** is generated to introduce slightly more randomness. If the shell compatibility level is set to 50 or lower, it reverts to the method from bash-5.0 and previous versions, so seeding the random number generator by assigning a value to **RANDOM** will produce the same sequence as in bash-5.0.
- If the command hash table is empty, bash versions prior to bash-5.1 printed an informational message to that effect, even when producing output that can be reused as input. Bash-5.1 suppresses that message when the **−l** option is supplied.

compat51

- The **unset** builtin treats attempts to unset array subscripts @ and * differently depending on whether the array is indexed or associative, and differently than in previous versions.
- Arithmetic commands (**((**...**))**) and the expressions in an arithmetic for statement can be expanded more than once.
- Expressions used as arguments to arithmetic operators in the **[[** conditional command can be expanded more than once.
- The expressions in substring parameter brace expansion can be expanded more than once.
- The expressions in the **$((**...**))** word expansion can be expanded more than once.
- Arithmetic expressions used as indexed array subscripts can be expanded more than once.
- **test −v**, when given an argument of **A[@]**, where **A** is an existing associative array, will return true if the array has any set elements. Bash-5.2 will look for and report on a key named @.
- The ${*parameter***[:]=***value*} word expansion will return *value*, before any variable-specific transformations have been performed (e.g., converting to lowercase). Bash-5.2 will return the final value assigned to the variable.

- Parsing command substitutions will behave as if extended globbing (see the description of the **shopt** builtin above) is enabled, so that parsing a command substitution containing an extglob pattern (say, as part of a shell function) will not fail. This assumes the intent is to enable extglob before the command is executed and word expansions are performed. It will fail at word expansion time if extglob hasn't been enabled by the time the command is executed.

compat52

- The **test** builtin uses its historical algorithm to parse parenthesized subexpressions when given five or more arguments.
- If the **−p** or **−P** option is supplied to the **bind** builtin, **bind** treats any arguments remaining after option processing as bindable command names, and displays any key sequences bound to those commands, instead of treating the arguments as key sequences to bind.

SEE ALSO

bash(1), sh(1)