Rocky Enterprise Linux 9.2 Manual Pages on command 'zshbuiltins.1'

C:\>man zshbuiltins.1

ZSHBUILTINS(1) General Commands Manual ZSHBUILTINS(1)

NAME

zshbuiltins - zsh built-in commands

SHELL BUILTIN COMMANDS

Some shell builtin commands take options as described in individual entries; these are often referred to in the list below as `flags' to avoid confusion with shell options, which may also have an effect on the behaviour of builtin commands. In this introductory section, `option' always has the meaning of an option to a com? mand that should be familiar to most command line users.

Typically, options are single letters preceded by a hyphen (-). Options that take an argument accept it either immediately following the option letter or after white space, for example `print -C3 {1..9}' or `print -C 3 {1..9}' are equivalent. Argu? ments to options are not the same as arguments to the command; the documentation indicates which is which. Options that do not take an argument may be combined in a single word, for example `print -rca -- *' and `print -r -c -a -- *' are equiva? lent.

Some shell builtin commands also take options that begin with `+' instead of `-'. The list below makes clear which commands these are.

Options (together with their individual arguments, if any) must appear in a group before any non-option arguments; once the first non-option argument has been found, option processing is terminated.

All builtin commands other than `echo' and precommand modifiers, even those that

have no options, can be given the argument `--' to terminate option processing. This indicates that the following words are non-option arguments, but is otherwise ignored. This is useful in cases where arguments to the command may begin with `-'. For historical reasons, most builtin commands (including `echo') also recog? nize a single `-' in a separate word for this purpose; note that this is less stan? dard and use of `--' is recommended.

- simple command

See the section `Precommand Modifiers' in zshmisc(1).

. file [arg ...]

Read commands from file and execute them in the current shell environment. If file does not contain a slash, or if PATH_DIRS is set, the shell looks in the components of $path to find the directory containing file. Files in the current directory are not read unless `.' appears somewhere in $path. If a file named `file.zwc' is found, is newer than file, and is the compiled form (created with the zcompile builtin) of file, then commands are read from that file instead of file.

If any arguments arg are given, they become the positional parameters; the old positional parameters are restored when the file is done executing. However, if no arguments are given, the positional parameters remain those of the calling context, and no restoring is done.

If file was not found the return status is 127; if file was found but con? tained a syntax error the return status is 126; else the return status is the exit status of the last command executed.

: [arg ...]

This command does nothing, although normal argument expansions is performed which may have effects on shell parameters. A zero exit status is returned.

alias [{+|-}gmrsL] [name[=value] ...]

For each name with a corresponding value, define an alias with that value. A trailing space in value causes the next word to be checked for alias ex? pansion. If the -g flag is present, define a global alias; global aliases are expanded even if they do not occur in command position.

If the -s flag is present, define a suffix alias: if the command word on a command line is in the form `text.name', where text is any non-empty string,

it is replaced by the text `value text.name'. Note that name is treated as a literal string, not a pattern. A trailing space in value is not special in this case. For example,

alias -s ps='gv --'

will cause the command `*.ps' to be expanded to `gv -- *.ps'. As alias ex? pansion is carried out earlier than globbing, the `*.ps' will then be ex? panded. Suffix aliases constitute a different name space from other aliases (so in the above example it is still possible to create an alias for the command ps) and the two sets are never listed together.

For each name with no value, print the value of name, if any. With no argu? ments, print all currently defined aliases other than suffix aliases. If the -m flag is given the arguments are taken as patterns (they should be quoted to preserve them from being interpreted as glob patterns), and the aliases matching these patterns are printed. When printing aliases and one of the -g, -r or -s flags is present, restrict the printing to global, regu? lar or suffix aliases, respectively; a regular alias is one which is neither a global nor a suffix alias. Using `+' instead of `-', or ending the op? tion list with a single `+', prevents the values of the aliases from being printed.

If the -L flag is present, then print each alias in a manner suitable for putting in a startup script. The exit status is nonzero if a name (with no value) is given for which no alias has been defined.

For more on aliases, include common problems, see the section ALIASING in zshmisc(1).

autoload [{+|-}RTUXdkmrtWz] [-w] [name ...]

See the section `Autoloading Functions' in zshmisc(1) for full details. The fpath parameter will be searched to find the function definition when the function is first referenced.

If name consists of an absolute path, the function is defined to load from the file given (searching as usual for dump files in the given location). The name of the function is the basename (non-directory part) of the file. It is normally an error if the function is not found in the given location; however, if the option -d is given, searching for the function defaults to

$fpath. If a function is loaded by absolute path, any functions loaded from it that are marked for autoload without an absolute path have the load path of the parent function temporarily prepended to $fpath.

If the option -r or -R is given, the function is searched for immediately and the location is recorded internally for use when the function is exe? cuted; a relative path is expanded using the value of $PWD. This protects against a change to $fpath after the call to autoload. With -r, if the function is not found, it is silently left unresolved until execution; with -R, an error message is printed and command processing aborted immediately the search fails, i.e. at the autoload command rather than at function exe? cution..

The flag -X may be used only inside a shell function. It causes the calling function to be marked for autoloading and then immediately loaded and exe? cuted, with the current array of positional parameters as arguments. This replaces the previous definition of the function. If no function definition is found, an error is printed and the function remains undefined and marked for autoloading. If an argument is given, it is used as a directory (i.e. it does not include the name of the function) in which the function is to be found; this may be combined with the -d option to allow the function search to default to $fpath if it is not in the given location.

The flag +X attempts to load each name as an autoloaded function, but does not execute it. The exit status is zero (success) if the function was not previously defined and a definition for it was found. This does not replace any existing definition of the function. The exit status is nonzero (fail? ure) if the function was already defined or when no definition was found. In the latter case the function remains undefined and marked for autoload? ing. If ksh-style autoloading is enabled, the function created will contain the contents of the file plus a call to the function itself appended to it, thus giving normal ksh autoloading behaviour on the first call to the func? tion. If the -m flag is also given each name is treated as a pattern and all functions already marked for autoload that match the pattern are loaded.

With the -t flag, turn on execution tracing; with -T, turn on execution tracing only for the current function, turning it off on entry to any called

functions that do not also have tracing enabled.

With the -U flag, alias expansion is suppressed when the function is loaded.

With the -w flag, the names are taken as names of files compiled with the zcompile builtin, and all functions defined in them are marked for autoload?ing.

The flags -z and -k mark the function to be autoloaded using the zsh or ksh style, as if the option KSH_AUTOLOAD were unset or were set, respectively. The flags override the setting of the option at the time the function is loaded.

Note that the autoload command makes no attempt to ensure the shell options set during the loading or execution of the file have any particular value. For this, the emulate command can be used:

    emulate zsh -c 'autoload -Uz func'

arranges that when func is loaded the shell is in native zsh emulation, and this emulation is also applied when func is run.

Some of the functions of autoload are also provided by functions -u or func?tions -U, but autoload is a more comprehensive interface.

bg [job ...]

job ... &

Put each specified job in the background, or the current job if none is specified.

bindkey

See the section `Zle Builtins' in zshzle(1).

break [n]

Exit from an enclosing for, while, until, select or repeat loop. If an arithmetic expression n is specified, then break n levels instead of just one.

builtin name [args ...]

Executes the builtin name, with the given args.

bye Same as exit.

cap See the section `The zsh/cap Module' in zshmodules(1).

cd [-qsLP] [arg]

cd [-qsLP] old new

cd [-qsLP] {+|-}n

Change the current directory. In the first form, change the current direc? tory to arg, or to the value of $HOME if arg is not specified. If arg is `-', change to the previous directory.

Otherwise, if arg begins with a slash, attempt to change to the directory given by arg.

If arg does not begin with a slash, the behaviour depends on whether the current directory `.' occurs in the list of directories contained in the shell parameter cdpath. If it does not, first attempt to change to the di? rectory arg under the current directory, and if that fails but cdpath is set and contains at least one element attempt to change to the directory arg un? der each component of cdpath in turn until successful. If `.' occurs in cd? path, then cdpath is searched strictly in order so that `.' is only tried at the appropriate point.

The order of testing cdpath is modified if the option POSIX_CD is set, as described in the documentation for the option.

If no directory is found, the option CDABLE_VARS is set, and a parameter named arg exists whose value begins with a slash, treat its value as the di? rectory. In that case, the parameter is added to the named directory hash table.

The second form of cd substitutes the string new for the string old in the name of the current directory, and tries to change to this new directory.

The third form of cd extracts an entry from the directory stack, and changes to that directory. An argument of the form `+n' identifies a stack entry by counting from the left of the list shown by the dirs command, starting with zero. An argument of the form `-n' counts from the right. If the PUSHD_MI? NUS option is set, the meanings of `+' and `-' in this context are swapped.

If the POSIX_CD option is set, this form of cd is not recognised and will be interpreted as the first form.

If the -q (quiet) option is specified, the hook function chpwd and the func? tions in the array chpwd_functions are not called. This is useful for calls to cd that do not change the environment seen by an interactive user.

If the -s option is specified, cd refuses to change the current directory if

the given pathname contains symlinks. If the -P option is given or the CHASE_LINKS option is set, symbolic links are resolved to their true values. If the -L option is given symbolic links are retained in the directory (and not resolved) regardless of the state of the CHASE_LINKS option.

chdir Same as cd.

clone See the section `The zsh/clone Module' in zshmodules(1).

command [-pvV] simple command

The simple command argument is taken as an external command instead of a function or builtin and is executed. If the POSIX_BUILTINS option is set, builtins will also be executed but certain special properties of them are suppressed. The -p flag causes a default path to be searched instead of that in $path. With the -v flag, command is similar to whence and with -V, it is equivalent to whence -v.

See also the section `Precommand Modifiers' in zshmisc(1).

comparguments

See the section `The zsh/computil Module' in zshmodules(1).

compcall

See the section `The zsh/compctl Module' in zshmodules(1).

compctl

See the section `The zsh/compctl Module' in zshmodules(1).

compdescribe

See the section `The zsh/computil Module' in zshmodules(1).

compfiles

See the section `The zsh/computil Module' in zshmodules(1).

compgroups

See the section `The zsh/computil Module' in zshmodules(1).

compquote

See the section `The zsh/computil Module' in zshmodules(1).

comptags

See the section `The zsh/computil Module' in zshmodules(1).

comptry

See the section `The zsh/computil Module' in zshmodules(1).

compvalues

See the section `The zsh/computil Module' in zshmodules(1).

continue [n]

Resume the next iteration of the enclosing for, while, until, select or re?peat loop. If an arithmetic expression n is specified, break out of n-1 loops and resume at the nth enclosing loop.

declare

Same as typeset.

dirs [-c] [arg ...]

dirs [-lpv]

With no arguments, print the contents of the directory stack. Directories are added to this stack with the pushd command, and removed with the cd or popd commands. If arguments are specified, load them onto the directory stack, replacing anything that was there, and push the current directory onto the stack.

-c clear the directory stack.

-l print directory names in full instead of using of using ~ expressions (see Dynamic and Static named directories in zshexpn(1)).

-p print directory entries one per line.

-v number the directories in the stack when printing.

disable [-afmprs] name ...

Temporarily disable the named hash table elements or patterns. The default is to disable builtin commands. This allows you to use an external command with the same name as a builtin command. The -a option causes disable to act on regular or global aliases. The -s option causes disable to act on suffix aliases. The -f option causes disable to act on shell functions. The -r options causes disable to act on reserved words. Without arguments all disabled hash table elements from the corresponding hash table are printed. With the -m flag the arguments are taken as patterns (which should be quoted to prevent them from undergoing filename expansion), and all hash table elements from the corresponding hash table matching these patterns are disabled. Disabled objects can be enabled with the enable command. With the option -p, name ... refer to elements of the shell's pattern syntax as described in the section `Filename Generation'. Certain elements can be

disabled separately, as given below.

Note that patterns not allowed by the current settings for the options EX? TENDED_GLOB, KSH_GLOB and SH_GLOB are never enabled, regardless of the set? ting here. For example, if EXTENDED_GLOB is not active, the pattern ^ is ineffective even if `disable -p "^"' has not been issued. The list below indicates any option settings that restrict the use of the pattern. It should be noted that setting SH_GLOB has a wider effect than merely dis? abling patterns as certain expressions, in particular those involving paren? theses, are parsed differently.

The following patterns may be disabled; all the strings need quoting on the command line to prevent them from being interpreted immediately as patterns and the patterns are shown below in single quotes as a reminder.

'?' The pattern character ? wherever it occurs, including when preceding a parenthesis with KSH_GLOB.

'*' The pattern character * wherever it occurs, including recursive glob? bing and when preceding a parenthesis with KSH_GLOB.

'[' Character classes.

'<' (NO_SH_GLOB)

Numeric ranges.

'|' (NO_SH_GLOB)

Alternation in grouped patterns, case statements, or KSH_GLOB paren? thesised expressions.

'(' (NO_SH_GLOB)

Grouping using single parentheses. Disabling this does not disable the use of parentheses for KSH_GLOB where they are introduced by a special character, nor for glob qualifiers (use `setopt NO_BARE_GLOB_QUAL' to disable glob qualifiers that use parentheses only).

'~' (EXTENDED_GLOB)

Exclusion in the form A~B.

'^' (EXTENDED_GLOB)

Exclusion in the form A^B.

'#' (EXTENDED_GLOB)

The pattern character # wherever it occurs, both for repetition of a previous pattern and for indicating globbing flags.

'?(' (KSH_GLOB)

The grouping form ?(...). Note this is also disabled if '?' is dis?abled.

'*(' (KSH_GLOB)

The grouping form *(...). Note this is also disabled if '*' is dis?abled.

'+(' (KSH_GLOB)

The grouping form +(...).

'!(' (KSH_GLOB)

The grouping form !(...).

'@(' (KSH_GLOB)

The grouping form @(...).

disown [job ...]

job ... &|

job ... &!

Remove the specified jobs from the job table; the shell will no longer re?port their status, and will not complain if you try to exit an interactive shell with them running or stopped. If no job is specified, disown the cur?rent job.

If the jobs are currently stopped and the AUTO_CONTINUE option is not set, a warning is printed containing information about how to make them running af?ter they have been disowned. If one of the latter two forms is used, the jobs will automatically be made running, independent of the setting of the AUTO_CONTINUE option.

echo [-neE] [arg ...]

Write each arg on the standard output, with a space separating each one. If the -n flag is not present, print a newline at the end. echo recognizes the following escape sequences:

\a bell character

\b backspace

\c suppress subsequent characters and final newline

\e escape

\f form feed

\n linefeed (newline)

\r carriage return

\t horizontal tab

\v vertical tab

\\ backslash

\0NNN character code in octal

\xNN character code in hexadecimal

\uNNNN unicode character code in hexadecimal

\UNNNNNNNN

unicode character code in hexadecimal

The -E flag, or the BSD_ECHO option, can be used to disable these escape se?

quences. In the latter case, -e flag can be used to enable them.

Note that for standards compliance a double dash does not terminate option

processing; instead, it is printed directly. However, a single dash does

terminate option processing, so the first dash, possibly following options,

is not printed, but everything following it is printed as an argument. The

single dash behaviour is different from other shells. For a more portable

way of printing text, see printf, and for a more controllable way of print?

ing text within zsh, see print.

echotc See the section `The zsh/termcap Module' in zshmodules(1).

echoti See the section `The zsh/terminfo Module' in zshmodules(1).

emulate [-lLR] [{zsh|sh|ksh|csh} [flags ...]]

Without any argument print current emulation mode.

With single argument set up zsh options to emulate the specified shell as

much as possible. csh will never be fully emulated. If the argument is not

one of the shells listed above, zsh will be used as a default; more pre?

cisely, the tests performed on the argument are the same as those used to

determine the emulation at startup based on the shell name, see the section

COMPATIBILITY in zsh(1) . In addition to setting shell options, the command

also restores the pristine state of pattern enables, as if all patterns had

been enabled using enable -p.

If the emulate command occurs inside a function that has been marked for ex?
ecution tracing with functions -t then the xtrace option will be turned on
regardless of emulation mode or other options. Note that code executed in?
side the function by the ., source, or eval commands is not considered to be
running directly from the function, hence does not provoke this behaviour.
If the -R switch is given, all settable options are reset to their default
value corresponding to the specified emulation mode, except for certain op?
tions describing the interactive environment; otherwise, only those options
likely to cause portability problems in scripts and functions are altered.
If the -L switch is given, the options LOCAL_OPTIONS, LOCAL_PATTERNS and LO?
CAL_TRAPS will be set as well, causing the effects of the emulate command
and any setopt, disable -p or enable -p, and trap commands to be local to
the immediately surrounding shell function, if any; normally these options
are turned off in all emulation modes except ksh. The -L switch is mutually
exclusive with the use of -c in flags.
If there is a single argument and the -l switch is given, the options that
would be set or unset (the latter indicated with the prefix `no') are
listed. -l can be combined with -L or -R and the list will be modified in
the appropriate way. Note the list does not depend on the current setting
of options, i.e. it includes all options that may in principle change, not
just those that would actually change.
The flags may be any of the invocation-time flags described in the section
INVOCATION in zsh(1), except that `-o EMACS' and `-o VI' may not be used.
Flags such as `+r'/`+o RESTRICTED' may be prohibited in some circumstances.
If -c arg appears in flags, arg is evaluated while the requested emulation
is temporarily in effect. In this case the emulation mode and all options
are restored to their previous values before emulate returns. The -R switch
may precede the name of the shell to emulate; note this has a meaning dis?
tinct from including -R in flags.
Use of -c enables `sticky' emulation mode for functions defined within the
evaluated expression: the emulation mode is associated thereafter with the
function so that whenever the function is executed the emulation (respecting
the -R switch, if present) and all options are set (and pattern disables

cleared) before entry to the function, and the state is restored after exit.

If the function is called when the sticky emulation is already in effect, either within an `emulate shell -c' expression or within another function with the same sticky emulation, entry and exit from the function do not cause options to be altered (except due to standard processing such as the LOCAL_OPTIONS option). This also applies to functions marked for autoload within the sticky emulation; the appropriate set of options will be applied at the point the function is loaded as well as when it is run.

For example:

```
emulate sh -c 'fni() { setopt cshnullglob; }
fno() { fni; }'
fno
```

The two functions fni and fno are defined with sticky sh emulation. fno is then executed, causing options associated with emulations to be set to their values in sh. fno then calls fni; because fni is also marked for sticky sh emulation, no option changes take place on entry to or exit from it. Hence the option cshnullglob, turned off by sh emulation, will be turned on within fni and remain on return to fno. On exit from fno, the emulation mode and all options will be restored to the state they were in before entry to the temporary emulation.

The documentation above is typically sufficient for the intended purpose of executing code designed for other shells in a suitable environment. More detailed rules follow.

1. The sticky emulation environment provided by `emulate shell -c' is identical to that provided by entry to a function marked for sticky emulation as a consequence of being defined in such an environment. Hence, for example, the sticky emulation is inherited by subfunctions defined within functions with sticky emulation.
2. No change of options takes place on entry to or exit from functions that are not marked for sticky emulation, other than those that would normally take place, even if those functions are called within sticky emulation.
3. No special handling is provided for functions marked for autoload nor

for functions present in wordcode created by the zcompile command.

4. The presence or absence of the -R switch to emulate corresponds to different sticky emulation modes, so for example `emulate sh -c', `emulate -R sh -c' and `emulate csh -c' are treated as three distinct sticky emulations.

5. Difference in shell options supplied in addition to the basic emula? tion also mean the sticky emulations are different, so for example `emulate zsh -c' and `emulate zsh -o cbases -c' are treated as dis? tinct sticky emulations.

enable [-afmprs] name ...

Enable the named hash table elements, presumably disabled earlier with dis? able. The default is to enable builtin commands. The -a option causes en? able to act on regular or global aliases. The -s option causes enable to act on suffix aliases. The -f option causes enable to act on shell func? tions. The -r option causes enable to act on reserved words. Without argu? ments all enabled hash table elements from the corresponding hash table are printed. With the -m flag the arguments are taken as patterns (should be quoted) and all hash table elements from the corresponding hash table match? ing these patterns are enabled. Enabled objects can be disabled with the disable builtin command.

enable -p reenables patterns disabled with disable -p. Note that it does not override globbing options; for example, `enable -p "~"' does not cause the pattern character ~ to be active unless the EXTENDED_GLOB option is also set. To enable all possible patterns (so that they may be individually dis? abled with disable -p), use `setopt EXTENDED_GLOB KSH_GLOB NO_SH_GLOB'.

eval [arg ...]

Read the arguments as input to the shell and execute the resulting com? mand(s) in the current shell process. The return status is the same as if the commands had been executed directly by the shell; if there are no args or they contain no commands (i.e. are an empty string or whitespace) the re? turn status is zero.

exec [-cl] [-a argv0] [command [arg ...]]

Replace the current shell with command rather than forking. If command is a

shell builtin command or a shell function, the shell executes it, and exits when the command is complete.

With -c clear the environment; with -l prepend - to the argv[0] string of the command executed (to simulate a login shell); with -a argv0 set the argv[0] string of the command executed. See the section `Precommand Modifiers' in zshmisc(1).

If the option POSIX_BUILTINS is set, command is never interpreted as a shell builtin command or shell function. This means further precommand modifiers such as builtin and noglob are also not interpreted within the shell. Hence command is always found by searching the command path.

If command is omitted but any redirections are specified, then the redirections will take effect in the current shell.

exit [n]

Exit the shell with the exit status specified by an arithmetic expression n; if none is specified, use the exit status from the last command executed.

An EOF condition will also cause the shell to exit, unless the IGNORE_EOF option is set.

See notes at the end of the section JOBS in zshmisc(1) for some possibly unexpected interactions of the exit command with jobs.

export [name[=value] ...]

The specified names are marked for automatic export to the environment of subsequently executed commands. Equivalent to typeset -gx. If a parameter specified does not already exist, it is created in the global scope.

false [arg ...]

Do nothing and return an exit status of 1.

fc [-e ename] [-LI] [-m match] [old=new ...] [first [last]]

fc -l [-LI] [-nrdfEiD] [-t timefmt] [-m match]
[old=new ...] [first [last]]

fc -p [-a] [filename [histsize [savehistsize]]]

fc -P

fc -ARWI [filename]

The fc command controls the interactive history mechanism. Note that reading and writing of history options is only performed if the shell is inter?

active. Usually this is detected automatically, but it can be forced by setting the interactive option when starting the shell.

The first two forms of this command select a range of events from first to last from the history list. The arguments first and last may be specified as a number or as a string. A negative number is used as an offset to the current history event number. A string specifies the most recent event be? ginning with the given string. All substitutions old=new, if any, are then performed on the text of the events.

In addition to the number range,

-I restricts to only internal events (not from $HISTFILE)

-L restricts to only local events (not from other shells, see SHARE_HIS? TORY in zshoptions(1) -- note that $HISTFILE is considered local when read at startup)

-m takes the first argument as a pattern (should be quoted) and only the history events matching this pattern are considered

If first is not specified, it will be set to -1 (the most recent event), or to -16 if the -l flag is given. If last is not specified, it will be set to first, or to -1 if the -l flag is given. However, if the current event has added entries to the history with `print -s' or `fc -R', then the default last for -l includes all new history entries since the current event began.

When the -l flag is given, the resulting events are listed on standard out? put. Otherwise the editor program specified by -e ename is invoked on a file containing these history events. If -e is not given, the value of the parameter FCEDIT is used; if that is not set the value of the parameter EDI? TOR is used; if that is not set a builtin default, usually `vi' is used. If ename is `-', no editor is invoked. When editing is complete, the edited command is executed.

The flag -r reverses the order of the events and the flag -n suppresses event numbers when listing.

Also when listing,

-d prints timestamps for each event

-f prints full time-date stamps in the US `MM/DD/YY hh:mm' format

-E prints full time-date stamps in the European `dd.mm.yyyy hh:mm' for?

mat

-i prints full time-date stamps in ISO8601 `yyyy-mm-dd hh:mm' format

-t fmt prints time and date stamps in the given format; fmt is formatted with the strftime function with the zsh extensions described for the %D{string} prompt format in the section EXPANSION OF PROMPT SEQUENCES in zshmisc(1). The resulting formatted string must be no more than 256 characters or will not be printed

-D prints elapsed times; may be combined with one of the options above

`fc -p' pushes the current history list onto a stack and switches to a new history list. If the -a option is also specified, this history list will be automatically popped when the current function scope is exited, which is a much better solution than creating a trap function to call `fc -P' manually. If no arguments are specified, the history list is left empty, $HISTFILE is unset, and $HISTSIZE & $SAVEHIST are set to their default values. If one argument is given, $HISTFILE is set to that filename, $HISTSIZE & $SAVEHIST are left unchanged, and the history file is read in (if it exists) to ini? tialize the new list. If a second argument is specified, $HISTSIZE & $SAVE? HIST are instead set to the single specified numeric value. Finally, if a third argument is specified, $SAVEHIST is set to a separate value from $HISTSIZE. You are free to change these environment values for the new his? tory list however you desire in order to manipulate the new history list.

`fc -P' pops the history list back to an older list saved by `fc -p'. The current list is saved to its $HISTFILE before it is destroyed (assuming that $HISTFILE and $SAVEHIST are set appropriately, of course). The values of $HISTFILE, $HISTSIZE, and $SAVEHIST are restored to the values they had when `fc -p' was called. Note that this restoration can conflict with making these variables "local", so your best bet is to avoid local declarations for these variables in functions that use `fc -p'. The one other guaran? teed-safe combination is declaring these variables to be local at the top of your function and using the automatic option (-a) with `fc -p'. Finally, note that it is legal to manually pop a push marked for automatic popping if you need to do so before the function exits.

`fc -R' reads the history from the given file, `fc -W' writes the history

out to the given file, and `fc -A' appends the history out to the given file. If no filename is specified, the $HISTFILE is assumed. If the -I option is added to -R, only those events that are not already contained within the internal history list are added. If the -I option is added to -A or -W, only those events that are new since last incremental append/write to the history file are appended/written. In any case, the created file will have no more than $SAVEHIST entries.

fg [job ...]

job ...

Bring each specified job in turn to the foreground. If no job is specified, resume the current job.

float [{+|-}Hghlprtux] [{+|-}EFLRZ [n]] [name[=value] ...]

Equivalent to typeset -E, except that options irrelevant to floating point numbers are not permitted.

functions [{+|-}UkmtTuWz] [-x num] [name ...]

functions -c oldfn newfn

functions -M [-s] mathfn [min [max [shellfn]]]

functions -M [-m pattern ...]

functions +M [-m] mathfn ...

Equivalent to typeset -f, with the exception of the -c, -x, -M and -W options. For functions -u and functions -U, see autoload, which provides additional options.

The -x option indicates that any functions output will have each leading tab for indentation, added by the shell to show syntactic structure, expanded to the given number num of spaces. num can also be 0 to suppress all indentation.

The -W option turns on the option WARN_NESTED_VAR for the named function or functions only. The option is turned off at the start of nested functions (apart from anonoymous functions) unless the called function also has the -W attribute.

The -c option causes oldfn to be copied to newfn. The copy is efficiently handled internally by reference counting. If oldfn was marked for autoload it is first loaded and if this fails the copy fails. Either function may

subsequently be redefined without affecting the other. A typical idiom is that oldfn is the name of a library shell function which is then redefined to call newfn, thereby installing a modified version of the function.

Use of the -M option may not be combined with any of the options handled by typeset -f.

functions -M mathfn defines mathfn as the name of a mathematical function recognised in all forms of arithmetical expressions; see the section `Arith? metic Evaluation' in zshmisc(1). By default mathfn may take any number of comma-separated arguments. If min is given, it must have exactly min args; if min and max are both given, it must have at least min and at most max args. max may be -1 to indicate that there is no upper limit.

By default the function is implemented by a shell function of the same name; if shellfn is specified it gives the name of the corresponding shell func? tion while mathfn remains the name used in arithmetical expressions. The name of the function in $0 is mathfn (not shellfn as would usually be the case), provided the option FUNCTION_ARGZERO is in effect. The positional parameters in the shell function correspond to the arguments of the mathe? matical function call. The result of the last arithmetical expression eval? uated inside the shell function (even if it is a form that normally only re? turns a status) gives the result of the mathematical function.

If the additional option -s is given to functions -M, the argument to the function is a single string: anything between the opening and matching clos? ing parenthesis is passed to the function as a single argument, even if it includes commas or white space. The minimum and maximum argument specifiers must therefore be 1 if given. An empty argument list is passed as a zero-length string.

functions -M with no arguments lists all such user-defined functions in the same form as a definition. With the additional option -m and a list of ar? guments, all functions whose mathfn matches one of the pattern arguments are listed.

function +M removes the list of mathematical functions; with the additional option -m the arguments are treated as patterns and all functions whose mathfn matches the pattern are removed. Note that the shell function imple?

menting the behaviour is not removed (regardless of whether its name coin? cides with mathfn).

For example, the following prints the cube of 3:

```
zmath_cube() { (( $1 * $1 * $1 )) }
functions -M cube 1 1 zmath_cube
print $(( cube(3) ))
```

The following string function takes a single argument, including the commas, so prints 11:

```
stringfn() { (( $#1 )) }
functions -Ms stringfn
print $(( stringfn(foo,bar,rod) ))
```

getcap See the section `The zsh/cap Module' in zshmodules(1).

getln [-AclneE] name ...

Read the top value from the buffer stack and put it in the shell parameter name. Equivalent to read -zr.

getopts optstring name [arg ...]

Checks the args for legal options. If the args are omitted, use the posi? tional parameters. A valid option argument begins with a `+' or a `-'. An argument not beginning with a `+' or a `-', or the argument `--', ends the options. Note that a single `-' is not considered a valid option argument. optstring contains the letters that getopts recognizes. If a letter is fol? lowed by a `:', that option requires an argument. The options can be sepa? rated from the argument by blanks.

Each time it is invoked, getopts places the option letter it finds in the shell parameter name, prepended with a `+' when arg begins with a `+'. The index of the next arg is stored in OPTIND. The option argument, if any, is stored in OPTARG.

The first option to be examined may be changed by explicitly assigning to OPTIND. OPTIND has an initial value of 1, and is normally set to 1 upon en? try to a shell function and restored upon exit (this is disabled by the POSIX_BUILTINS option). OPTARG is not reset and retains its value from the most recent call to getopts. If either of OPTIND or OPTARG is explicitly unset, it remains unset, and the index or option argument is not stored.

The option itself is still stored in name in this case.

A leading `:' in optstring causes getopts to store the letter of any invalid option in OPTARG, and to set name to `?' for an unknown option and to `:' when a required argument is missing. Otherwise, getopts sets name to `?' and prints an error message when an option is invalid. The exit status is nonzero when there are no more options.

hash [-Ldfmrv] [name[=value]] ...

hash can be used to directly modify the contents of the command hash table, and the named directory hash table. Normally one would modify these tables by modifying one's PATH (for the command hash table) or by creating appro? priate shell parameters (for the named directory hash table). The choice of hash table to work on is determined by the -d option; without the option the command hash table is used, and with the option the named directory hash ta? ble is used.

A command name starting with a / is never hashed, whether by explicit use of the hash command or otherwise. Such a command is always found by direct look up in the file system.

Given no arguments, and neither the -r or -f options, the selected hash ta? ble will be listed in full.

The -r option causes the selected hash table to be emptied. It will be sub? sequently rebuilt in the normal fashion. The -f option causes the selected hash table to be fully rebuilt immediately. For the command hash table this hashes all the absolute directories in the PATH, and for the named directory hash table this adds all users' home directories. These two options cannot be used with any arguments.

The -m option causes the arguments to be taken as patterns (which should be quoted) and the elements of the hash table matching those patterns are printed. This is the only way to display a limited selection of hash table elements.

For each name with a corresponding value, put `name' in the selected hash table, associating it with the pathname `value'. In the command hash table, this means that whenever `name' is used as a command argument, the shell will try to execute the file given by `value'. In the named directory hash

table, this means that `value' may be referred to as `~name'.

For each name with no corresponding value, attempt to add name to the hash table, checking what the appropriate value is in the normal manner for that hash table. If an appropriate value can't be found, then the hash table will be unchanged.

The -v option causes hash table entries to be listed as they are added by explicit specification. If has no effect if used with -f.

If the -L flag is present, then each hash table entry is printed in the form of a call to hash.

history

Same as fc -l.

integer [{+|-}Hghlprtux] [{+|-}LRZi [n]] [name[=value] ...]

Equivalent to typeset -i, except that options irrelevant to integers are not permitted.

jobs [-dlprs] [job ...]

jobs -Z string

Lists information about each given job, or all jobs if job is omitted. The -l flag lists process IDs, and the -p flag lists process groups. If the -r flag is specified only running jobs will be listed and if the -s flag is given only stopped jobs are shown. If the -d flag is given, the directory from which the job was started (which may not be the current directory of the job) will also be shown.

The -Z option replaces the shell's argument and environment space with the given string, truncated if necessary to fit. This will normally be visible in ps (ps(1)) listings. This feature is typically used by daemons, to indi? cate their state.

kill [-s signal_name | -n signal_number | -sig] job ...

kill -l [sig ...]

Sends either SIGTERM or the specified signal to the given jobs or processes. Signals are given by number or by names, with or without the `SIG' prefix. If the signal being sent is not `KILL' or `CONT', then the job will be sent a `CONT' signal if it is stopped. The argument job can be the process ID of a job not in the job list. In the second form, kill -l, if sig is not spec?

ified the signal names are listed. Otherwise, for each sig that is a name, the corresponding signal number is listed. For each sig that is a signal number or a number representing the exit status of a process which was ter? minated or stopped by a signal the name of the signal is printed.

On some systems, alternative signal names are allowed for a few signals. Typical examples are SIGCHLD and SIGCLD or SIGPOLL and SIGIO, assuming they correspond to the same signal number. kill -l will only list the preferred form, however kill -l alt will show if the alternative form corresponds to a signal number. For example, under Linux kill -l IO and kill -l POLL both output 29, hence kill -IO and kill -POLL have the same effect.

Many systems will allow process IDs to be negative to kill a process group or zero to kill the current process group.

let arg ...

Evaluate each arg as an arithmetic expression. See the section `Arithmetic Evaluation' in zshmisc(1) for a description of arithmetic expressions. The exit status is 0 if the value of the last expression is nonzero, 1 if it is zero, and 2 if an error occurred.

limit [-hs] [resource [limit]] ...

Set or display resource limits. Unless the -s flag is given, the limit ap? plies only the children of the shell. If -s is given without other argu? ments, the resource limits of the current shell is set to the previously set resource limits of the children.

If limit is not specified, print the current limit placed on resource, oth? erwise set the limit to the specified value. If the -h flag is given, use hard limits instead of soft limits. If no resource is given, print all lim? its.

When looping over multiple resources, the shell will abort immediately if it detects a badly formed argument. However, if it fails to set a limit for some other reason it will continue trying to set the remaining limits.

resource can be one of:

addressspace

Maximum amount of address space used.

aiomemorylocked

Maximum amount of memory locked in RAM for AIO operations.

aiooperations

Maximum number of AIO operations.

cachedthreads

Maximum number of cached threads.

coredumpsize

Maximum size of a core dump.

cputime

Maximum CPU seconds per process.

datasize

Maximum data size (including stack) for each process.

descriptors

Maximum value for a file descriptor.

filesize

Largest single file allowed.

kqueues

Maximum number of kqueues allocated.

maxproc

Maximum number of processes.

maxpthreads

Maximum number of threads per process.

memorylocked

Maximum amount of memory locked in RAM.

memoryuse

Maximum resident set size.

msgqueue

Maximum number of bytes in POSIX message queues.

posixlocks

Maximum number of POSIX locks per user.

pseudoterminals

Maximum number of pseudo-terminals.

resident

Maximum resident set size.

sigpending

Maximum number of pending signals.

sockbufsize

Maximum size of all socket buffers.

stacksize

Maximum stack size for each process.

swapsize

Maximum amount of swap used.

vmemorysize

Maximum amount of virtual memory.

Which of these resource limits are available depends on the system. re?

source can be abbreviated to any unambiguous prefix. It can also be an in?

teger, which corresponds to the integer defined for the resource by the op?

erating system.

If argument corresponds to a number which is out of the range of the re?

sources configured into the shell, the shell will try to read or write the

limit anyway, and will report an error if this fails. As the shell does not

store such resources internally, an attempt to set the limit will fail un?

less the -s option is present.

limit is a number, with an optional scaling factor, as follows:

nh hours

nk kilobytes (default)

nm megabytes or minutes

ng gigabytes

[mm:]ss

minutes and seconds

The limit command is not made available by default when the shell starts in

a mode emulating another shell. It can be made available with the command

`zmodload -F zsh/rlimits b:limit'.

local [{+|-}AHUahlprtux] [{+|-}EFLRZi [n]] [name[=value] ...]

Same as typeset, except that the options -g, and -f are not permitted. In

this case the -x option does not force the use of -g, i.e. exported vari?

ables will be local to functions.

log List all users currently logged in who are affected by the current setting of the watch parameter.

logout [n]

Same as exit, except that it only works in a login shell.

noglob simple command

See the section `Precommand Modifiers' in zshmisc(1).

popd [-q] [{+|-}n]

Remove an entry from the directory stack, and perform a cd to the new top directory. With no argument, the current top entry is removed. An argument of the form `+n' identifies a stack entry by counting from the left of the list shown by the dirs command, starting with zero. An argument of the form -n counts from the right. If the PUSHD_MINUS option is set, the meanings of `+' and `-' in this context are swapped.

If the -q (quiet) option is specified, the hook function chpwd and the func? tions in the array $chpwd_functions are not called, and the new directory stack is not printed. This is useful for calls to popd that do not change the environment seen by an interactive user.

print [-abcDilmnNoOpPrsSz] [-u n] [-f format] [-C cols]
[-v name] [-xX tabstop] [-R [-en]] [arg ...]

With the `-f' option the arguments are printed as described by printf. With no flags or with the flag `-', the arguments are printed on the standard output as described by echo, with the following differences: the escape se? quence `\M-x' (or `\Mx') metafies the character x (sets the highest bit), `\C-x' (or `\Cx') produces a control character (`\C-@' and `\C-?' give the characters NULL and delete), a character code in octal is represented by `\NNN' (instead of `\0NNN'), and `\E' is a synonym for `\e'. Finally, if not in an escape sequence, `\' escapes the following character and is not printed.

-a Print arguments with the column incrementing first. Only useful with the -c and -C options.

-b Recognize all the escape sequences defined for the bindkey command, see the section `Zle Builtins' in zshzle(1).

-c Print the arguments in columns. Unless -a is also given, arguments

are printed with the row incrementing first.

-C cols

Print the arguments in cols columns. Unless -a is also given, argu?
ments are printed with the row incrementing first.

-D Treat the arguments as paths, replacing directory prefixes with ~ ex?
pressions corresponding to directory names, as appropriate.

-i If given together with -o or -O, sorting is performed case-indepen?
dently.

-l Print the arguments separated by newlines instead of spaces. Note:
if the list of arguments is empty, print -l will still output one
empty line. To print a possibly-empty list of arguments one per line,
use print -C1, as in `print -rC1 -- "$list[@]"'.

-m Take the first argument as a pattern (should be quoted), and remove
it from the argument list together with subsequent arguments that do
not match this pattern.

-n Do not add a newline to the output.

-N Print the arguments separated and terminated by nulls. Again, print
-rNC1 -- "$list[@]" is a canonical way to print an arbitrary list as
null-delimited records.

-o Print the arguments sorted in ascending order.

-O Print the arguments sorted in descending order.

-p Print the arguments to the input of the coprocess.

-P Perform prompt expansion (see EXPANSION OF PROMPT SEQUENCES in zsh?
misc(1)). In combination with `-f', prompt escape sequences are
parsed only within interpolated arguments, not within the format
string.

-r Ignore the escape conventions of echo.

-R Emulate the BSD echo command, which does not process escape sequences
unless the -e flag is given. The -n flag suppresses the trailing
newline. Only the -e and -n flags are recognized after -R; all other
arguments and options are printed.

-s Place the results in the history list instead of on the standard out?
put. Each argument to the print command is treated as a single word

in the history, regardless of its content.

-S Place the results in the history list instead of on the standard out?

put. In this case only a single argument is allowed; it will be split into words as if it were a full shell command line. The effect is similar to reading the line from a history file with the HIST_LEX_WORDS option active.

-u n Print the arguments to file descriptor n.

-v name

Store the printed arguments as the value of the parameter name.

-x tab-stop

Expand leading tabs on each line of output in the printed string as? suming a tab stop every tab-stop characters. This is appropriate for formatting code that may be indented with tabs. Note that leading tabs of any argument to print, not just the first, are expanded, even if print is using spaces to separate arguments (the column count is maintained across arguments but may be incorrect on output owing to previous unexpanded tabs).

The start of the output of each print command is assumed to be aligned with a tab stop. Widths of multibyte characters are handled if the option MULTIBYTE is in effect. This option is ignored if other formatting options are in effect, namely column alignment or printf style, or if output is to a special location such as shell history or the command line editor.

-X tab-stop

This is similar to -x, except that all tabs in the printed string are expanded. This is appropriate if tabs in the arguments are being used to produce a table format.

-z Push the arguments onto the editing buffer stack, separated by spa? ces.

If any of `-m', `-o' or `-O' are used in combination with `-f' and there are no arguments (after the removal process in the case of `-m') then nothing is printed.

Print the arguments according to the format specification. Formatting rules are the same as used in C. The same escape sequences as for echo are recog? nised in the format. All C conversion specifications ending in one of cs? diouxXeEfgGn are handled. In addition to this, `%b' can be used instead of `%s' to cause escape sequences in the argument to be recognised and `%q' can be used to quote the argument in such a way that allows it to be reused as shell input. With the numeric format specifiers, if the corresponding argu? ment starts with a quote character, the numeric value of the following char? acter is used as the number to print; otherwise the argument is evaluated as an arithmetic expression. See the section `Arithmetic Evaluation' in zsh? misc(1) for a description of arithmetic expressions. With `%n', the corre? sponding argument is taken as an identifier which is created as an integer parameter.

Normally, conversion specifications are applied to each argument in order but they can explicitly specify the nth argument is to be used by replacing `%' by `%n$' and `*' by `*n$'. It is recommended that you do not mix refer? ences of this explicit style with the normal style and the handling of such mixed styles may be subject to future change.

If arguments remain unused after formatting, the format string is reused un? til all arguments have been consumed. With the print builtin, this can be suppressed by using the -r option. If more arguments are required by the format than have been specified, the behaviour is as if zero or an empty string had been specified as the argument.

The -v option causes the output to be stored as the value of the parameter name, instead of printed. If name is an array and the format string is reused when consuming arguments then one array element will be used for each use of the format string.

pushd [-qsLP] [arg]

pushd [-qsLP] old new

pushd [-qsLP] {+|-}n

Change the current directory, and push the old current directory onto the directory stack. In the first form, change the current directory to arg.

If arg is not specified, change to the second directory on the stack (that

is, exchange the top two entries), or change to $HOME if the PUSHD_TO_HOME option is set or if there is only one entry on the stack. Otherwise, arg is interpreted as it would be by cd. The meaning of old and new in the second form is also the same as for cd.

The third form of pushd changes directory by rotating the directory list. An argument of the form `+n' identifies a stack entry by counting from the left of the list shown by the dirs command, starting with zero. An argument of the form `-n' counts from the right. If the PUSHD_MINUS option is set, the meanings of `+' and `-' in this context are swapped.

If the -q (quiet) option is specified, the hook function chpwd and the func? tions in the array $chpwd_functions are not called, and the new directory stack is not printed. This is useful for calls to pushd that do not change the environment seen by an interactive user.

If the option -q is not specified and the shell option PUSHD_SILENT is not set, the directory stack will be printed after a pushd is performed.

The options -s, -L and -P have the same meanings as for the cd builtin.

pushln [arg ...]

Equivalent to print -nz.

pwd [-rLP]

Print the absolute pathname of the current working directory. If the -r or the -P flag is specified, or the CHASE_LINKS option is set and the -L flag is not given, the printed path will not contain symbolic links.

r Same as fc -e -.

read [-rszpqAclneE] [-t [num]] [-k [num]] [-d delim]

[-u n] [name[?prompt]] [name ...]

Read one line and break it into fields using the characters in $IFS as sepa? rators, except as noted below. The first field is assigned to the first name, the second field to the second name, etc., with leftover fields as? signed to the last name. If name is omitted then REPLY is used for scalars and reply for arrays.

-r Raw mode: a `\' at the end of a line does not signify line continua? tion and backslashes in the line don't quote the following character and are not removed.

-s Don't echo back characters if reading from the terminal.

-q Read only one character from the terminal and set name to `y' if this character was `y' or `Y' and to `n' otherwise. With this flag set the return status is zero only if the character was `y' or `Y'. This option may be used with a timeout (see -t); if the read times out, or encounters end of file, status 2 is returned. Input is read from the terminal unless one of -u or -p is present. This option may also be used within zle widgets.

-k [num]

Read only one (or num) characters. All are assigned to the first name, without word splitting. This flag is ignored when -q is present. Input is read from the terminal unless one of -u or -p is present. This option may also be used within zle widgets.

Note that despite the mnemonic `key' this option does read full char? acters, which may consist of multiple bytes if the option MULTIBYTE is set.

-z Read one entry from the editor buffer stack and assign it to the first name, without word splitting. Text is pushed onto the stack with `print -z' or with push-line from the line editor (see zsh? zle(1)). This flag is ignored when the -k or -q flags are present.

-e

-E The input read is printed (echoed) to the standard output. If the -e flag is used, no input is assigned to the parameters.

-A The first name is taken as the name of an array and all words are as? signed to it.

-c

-l These flags are allowed only if called inside a function used for completion (specified with the -K flag to compctl). If the -c flag is given, the words of the current command are read. If the -l flag is given, the whole line is assigned as a scalar. If both flags are present, -l is used and -c is ignored.

-n Together with -c, the number of the word the cursor is on is read. With -l, the index of the character the cursor is on is read. Note

that the command name is word number 1, not word 0, and that when the cursor is at the end of the line, its character index is the length of the line plus one.

-u n Input is read from file descriptor n.

-p Input is read from the coprocess.

-d delim

Input is terminated by the first character of delim instead of by newline.

-t [num]

Test if input is available before attempting to read. If num is present, it must begin with a digit and will be evaluated to give a number of seconds, which may be a floating point number; in this case the read times out if input is not available within this time. If num is not present, it is taken to be zero, so that read returns im? mediately if no input is available. If no input is available, return status 1 and do not set any variables.

This option is not available when reading from the editor buffer with -z, when called from within completion with -c or -l, with -q which clears the input queue before reading, or within zle where other mechanisms should be used to test for input.

Note that read does not attempt to alter the input processing mode. The default mode is canonical input, in which an entire line is read at a time, so usually `read -t' will not read anything until an en? tire line has been typed. However, when reading from the terminal with -k input is processed one key at a time; in this case, only availability of the first character is tested, so that e.g. `read -t -k 2' can still block on the second character. Use two instances of `read -t -k' if this is not what is wanted.

If the first argument contains a `?', the remainder of this word is used as a prompt on standard error when the shell is interactive.

The value (exit status) of read is 1 when an end-of-file is encountered, or when -c or -l is present and the command is not called from a compctl func? tion, or as described for -q. Otherwise the value is 0.

The behavior of some combinations of the -k, -p, -q, -u and -z flags is un?
defined. Presently -q cancels all the others, -p cancels -u, -k cancels -z,
and otherwise -z cancels both -p and -u.
The -c or -l flags cancel any and all of -kpquz.

readonly
Same as typeset -r. With the POSIX_BUILTINS option set, same as typeset
-gr.

rehash Same as hash -r.

return [n]
Causes a shell function or `.' script to return to the invoking script with
the return status specified by an arithmetic expression n. If n is omitted,
the return status is that of the last command executed.
If return was executed from a trap in a TRAPNAL function, the effect is dif?
ferent for zero and non-zero return status. With zero status (or after an
implicit return at the end of the trap), the shell will return to whatever
it was previously processing; with a non-zero status, the shell will behave
as interrupted except that the return status of the trap is retained. Note
that the numeric value of the signal which caused the trap is passed as the
first argument, so the statement `return $((128+$1))' will return the same
status as if the signal had not been trapped.

sched See the section `The zsh/sched Module' in zshmodules(1).

set [{+|-}options | {+|-}o [option_name]] ... [{+|-}A [name]]
[arg ...]
Set the options for the shell and/or set the positional parameters, or de?
clare and set an array. If the -s option is given, it causes the specified
arguments to be sorted before assigning them to the positional parameters
(or to the array name if -A is used). With +s sort arguments in descending
order. For the meaning of the other flags, see zshoptions(1). Flags may be
specified by name using the -o option. If no option name is supplied with
-o, the current option states are printed: see the description of setopt
below for more information on the format. With +o they are printed in a
form that can be used as input to the shell.
If the -A flag is specified, name is set to an array containing the given

args; if no name is specified, all arrays are printed together with their values.

If +A is used and name is an array, the given arguments will replace the initial elements of that array; if no name is specified, all arrays are printed without their values.

The behaviour of arguments after -A name or +A name depends on whether the option KSH_ARRAYS is set. If it is not set, all arguments following name are treated as values for the array, regardless of their form. If the op? tion is set, normal option processing continues at that point; only regular arguments are treated as values for the array. This means that

set -A array -x -- foo

sets array to `-x -- foo' if KSH_ARRAYS is not set, but sets the array to foo and turns on the option `-x' if it is set.

If the -A flag is not present, but there are arguments beyond the options, the positional parameters are set. If the option list (if any) is termi? nated by `--', and there are no further arguments, the positional parameters will be unset.

If no arguments and no `--' are given, then the names and values of all pa? rameters are printed on the standard output. If the only argument is `+', the names of all parameters are printed.

For historical reasons, `set -' is treated as `set +xv' and `set - args' as `set +xv -- args' when in any other emulation mode than zsh's native mode.

setcap See the section `The zsh/cap Module' in zshmodules(1).

setopt [{+|-}options | {+|-}o option_name] [-m] [name ...]

Set the options for the shell. All options specified either with flags or by name are set.

If no arguments are supplied, the names of all options currently set are printed. The form is chosen so as to minimize the differences from the de? fault options for the current emulation (the default emulation being native zsh, shown as <Z> in zshoptions(1)). Options that are on by default for the emulation are shown with the prefix no only if they are off, while other op? tions are shown without the prefix no and only if they are on. In addition to options changed from the default state by the user, any options activated

automatically by the shell (for example, SHIN_STDIN or INTERACTIVE) will be shown in the list. The format is further modified by the option KSH_OP? TION_PRINT, however the rationale for choosing options with or without the no prefix remains the same in this case.

If the -m flag is given the arguments are taken as patterns (which should be quoted to protect them from filename expansion), and all options with names matching these patterns are set.

Note that a bad option name does not cause execution of subsequent shell code to be aborted; this is behaviour is different from that of `set -o'.

This is because set is regarded as a special builtin by the POSIX standard, but setopt is not.

shift [-p] [n] [name ...]

The positional parameters ${n+1} ... are renamed to $1 ..., where n is an arithmetic expression that defaults to 1. If any names are given then the arrays with these names are shifted instead of the positional parameters.

If the option -p is given arguments are instead removed (popped) from the end rather than the start of the array.

source file [arg ...]

Same as `.', except that the current directory is always searched and is al? ways searched first, before directories in $path.

stat See the section `The zsh/stat Module' in zshmodules(1).

suspend [-f]

Suspend the execution of the shell (send it a SIGTSTP) until it receives a SIGCONT. Unless the -f option is given, this will refuse to suspend a login shell.

test [arg ...]

[[arg ...]]

Like the system version of test. Added for compatibility; use conditional expressions instead (see the section `Conditional Expressions'). The main differences between the conditional expression syntax and the test and [builtins are: these commands are not handled syntactically, so for example an empty variable expansion may cause an argument to be omitted; syntax er? rors cause status 2 to be returned instead of a shell error; and arithmetic

operators expect integer arguments rather than arithmetic expressions.

The command attempts to implement POSIX and its extensions where these are specified. Unfortunately there are intrinsic ambiguities in the syntax; in particular there is no distinction between test operators and strings that resemble them. The standard attempts to resolve these for small numbers of arguments (up to four); for five or more arguments compatibility cannot be relied on. Users are urged wherever possible to use the `[[' test syntax which does not have these ambiguities.

times Print the accumulated user and system times for the shell and for processes run from the shell.

trap [arg] [sig ...]

arg is a series of commands (usually quoted to protect it from immediate evaluation by the shell) to be read and executed when the shell receives any of the signals specified by one or more sig args. Each sig can be given as a number, or as the name of a signal either with or without the string SIG in front (e.g. 1, HUP, and SIGHUP are all the same signal).

If arg is `-', then the specified signals are reset to their defaults, or, if no sig args are present, all traps are reset.

If arg is an empty string, then the specified signals are ignored by the shell (and by the commands it invokes).

If arg is omitted but one or more sig args are provided (i.e. the first ar? gument is a valid signal number or name), the effect is the same as if arg had been specified as `-'.

The trap command with no arguments prints a list of commands associated with each signal.

If sig is ZERR then arg will be executed after each command with a nonzero exit status. ERR is an alias for ZERR on systems that have no SIGERR signal (this is the usual case).

If sig is DEBUG then arg will be executed before each command if the option DEBUG_BEFORE_CMD is set (as it is by default), else after each command.

Here, a `command' is what is described as a `sublist' in the shell grammar, see the section SIMPLE COMMANDS & PIPELINES in zshmisc(1). If DEBUG_BE? FORE_CMD is set various additional features are available. First, it is

possible to skip the next command by setting the option ERR_EXIT; see the description of the ERR_EXIT option in zshoptions(1). Also, the shell param? eter ZSH_DEBUG_CMD is set to the string corresponding to the command to be executed following the trap. Note that this string is reconstructed from the internal format and may not be formatted the same way as the original text. The parameter is unset after the trap is executed.

If sig is 0 or EXIT and the trap statement is executed inside the body of a function, then the command arg is executed after the function completes. The value of $? at the start of execution is the exit status of the shell or the return status of the function exiting. If sig is 0 or EXIT and the trap statement is not executed inside the body of a function, then the command arg is executed when the shell terminates; the trap runs before any zshexit hook functions.

ZERR, DEBUG, and EXIT traps are not executed inside other traps. ZERR and DEBUG traps are kept within subshells, while other traps are reset.

Note that traps defined with the trap builtin are slightly different from those defined as `TRAPNAL () { ... }', as the latter have their own function environment (line numbers, local variables, etc.) while the former use the environment of the command in which they were called. For example,

```
trap 'print $LINENO' DEBUG
```

will print the line number of a command executed after it has run, while

```
TRAPDEBUG() { print $LINENO; }
```

will always print the number zero.

Alternative signal names are allowed as described under kill above. Defin? ing a trap under either name causes any trap under an alternative name to be removed. However, it is recommended that for consistency users stick exclu? sively to one name or another.

true [arg ...]

Do nothing and return an exit status of 0.

ttyctl [-fu]

The -f option freezes the tty (i.e. terminal or terminal emulator), and -u unfreezes it. When the tty is frozen, no changes made to the tty settings by external programs will be honored by the shell, except for changes in the

size of the screen; the shell will simply reset the settings to their previ?
ous values as soon as each command exits or is suspended. Thus, stty and
similar programs have no effect when the tty is frozen. Freezing the tty
does not cause the current state to be remembered: instead, it causes future
changes to the state to be blocked.
Without options it reports whether the terminal is frozen or not.
Note that, regardless of whether the tty is frozen or not, the shell needs
to change the settings when the line editor starts, so unfreezing the tty
does not guarantee settings made on the command line are preserved. Strings
of commands run between editing the command line will see a consistent tty
state. See also the shell variable STTY for a means of initialising the tty
before running external commands.

type [-wfpamsS] name ...

Equivalent to whence -v.

typeset [{+|-}AHUaghlmrtux] [{+|-}EFLRZip [n]]
[+] [name[=value] ...]

typeset -T [{+|-}Uglrux] [{+|-}LRZp [n]]
[+ | SCALAR[=value] array[=(value ...)] [sep]]

typeset -f [{+|-}TUkmtuz] [+] [name ...]

Set or display attributes and values for shell parameters.
Except as noted below for control flags that change the behavior, a parame?
ter is created for each name that does not already refer to one. When in?
side a function, a new parameter is created for every name (even those that
already exist), and is unset again when the function completes. See `Local
Parameters' in zshparam(1). The same rules apply to special shell parame?
ters, which retain their special attributes when made local.
For each name=value assignment, the parameter name is set to value.
If the shell option TYPESET_SILENT is not set, for each remaining name that
refers to a parameter that is already set, the name and value of the parame?
ter are printed in the form of an assignment. Nothing is printed for
newly-created parameters, or when any attribute flags listed below are given
along with the name. Using `+' instead of minus to introduce an attribute
turns it off.

If no name is present, the names and values of all parameters are printed. In this case the attribute flags restrict the display to only those parame? ters that have the specified attributes, and using `+' rather than `-' to introduce the flag suppresses printing of the values of parameters when there is no parameter name.

All forms of the command handle scalar assignment. Array assignment is pos? sible if any of the reserved words declare, export, float, integer, local, readonly or typeset is matched when the line is parsed (N.B. not when it is executed). In this case the arguments are parsed as assignments, except that the `+=' syntax and the GLOB_ASSIGN option are not supported, and scalar values after = are not split further into words, even if expanded (regardless of the setting of the KSH_TYPESET option; this option is obso? lete).

Examples of the differences between command and reserved word parsing:

```
# Reserved word parsing
typeset svar=$(echo one word) avar=(several words)
```

The above creates a scalar parameter svar and an array parameter avar as if the assignments had been

```
svar="one word"
avar=(several words)
```

On the other hand:

```
# Normal builtin interface
builtin typeset svar=$(echo two words)
```

The builtin keyword causes the above to use the standard builtin interface to typeset in which argument parsing is performed in the same way as for other commands. This example creates a scalar svar containing the value two and another scalar parameter words with no value. An array value in this case would either cause an error or be treated as an obscure set of glob qualifiers.

Arbitrary arguments are allowed if they take the form of assignments after command line expansion; however, these only perform scalar assignment:

```
var='svar=val'
typeset $var
```

The above sets the scalar parameter svar to the value val. Parentheses around the value within var would not cause array assignment as they will be treated as ordinary characters when $var is substituted. Any non-trivial expansion in the name part of the assignment causes the argument to be treated in this fashion:

typeset {var1,var2,var3}=name

The above syntax is valid, and has the expected effect of setting the three parameters to the same value, but the command line is parsed as a set of three normal command line arguments to typeset after expansion. Hence it is not possible to assign to multiple arrays by this means.

Note that each interface to any of the commands my be disabled separately. For example, `disable -r typeset' disables the reserved word interface to typeset, exposing the builtin interface, while `disable typeset' disables the builtin. Note that disabling the reserved word interface for typeset may cause problems with the output of `typeset -p', which assumes the re? served word interface is available in order to restore array and associative array values.

Unlike parameter assignment statements, typeset's exit status on an assign? ment that involves a command substitution does not reflect the exit status of the command substitution. Therefore, to test for an error in a command substitution, separate the declaration of the parameter from its initializa? tion:

```
# WRONG
typeset var1=$(exit 1) || echo "Trouble with var1"
# RIGHT
typeset var1 && var1=$(exit 1) || echo "Trouble with var1"
```

To initialize a parameter param to a command output and mark it readonly, use typeset -r param or readonly param after the parameter assignment state? ment.

If no attribute flags are given, and either no name arguments are present or the flag +m is used, then each parameter name printed is preceded by a list of the attributes of that parameter (array, association, exported, float, integer, readonly, or undefined for autoloaded parameters not yet loaded).

If +m is used with attribute flags, and all those flags are introduced with +, the matching parameter names are printed but their values are not.

The following control flags change the behavior of typeset:

+ If `+' appears by itself in a separate word as the last option, then the names of all parameters (functions with -f) are printed, but the values (function bodies) are not. No name arguments may appear, and it is an error for any other options to follow `+'. The effect of `+' is as if all attribute flags which precede it were given with a `+' prefix. For example, `typeset -U +' is equivalent to `typeset +U' and displays the names of all arrays having the uniqueness attri? bute, whereas `typeset -f -U +' displays the names of all autoload? able functions. If + is the only option, then type information (ar? ray, readonly, etc.) is also printed for each parameter, in the same manner as `typeset +m "*"'.

-g The -g (global) means that any resulting parameter will not be re? stricted to local scope. Note that this does not necessarily mean that the parameter will be global, as the flag will apply to any ex? isting parameter (even if unset) from an enclosing function. This flag does not affect the parameter after creation, hence it has no effect when listing existing parameters, nor does the flag +g have any effect except in combination with -m (see below).

-m If the -m flag is given the name arguments are taken as patterns (use quoting to prevent these from being interpreted as file patterns). With no attribute flags, all parameters (or functions with the -f flag) with matching names are printed (the shell option TYPE? SET_SILENT is not used in this case).

If the +g flag is combined with -m, a new local parameter is created for every matching parameter that is not already local. Otherwise -m applies all other flags or assignments to the existing parameters.

Except when assignments are made with name=value, using +m forces the matching parameters and their attributes to be printed, even inside a function. Note that -m is ignored if no patterns are given, so `typeset -m' displays attributes but `typeset -a +m' does not.

-p [n]

If the -p option is given, parameters and values are printed in the form of a typeset command with an assignment, regardless of other flags and options. Note that the -H flag on parameters is respected; no value will be shown for these parameters.

-p may be followed by an optional integer argument. Currently only the value 1 is supported. In this case arrays and associative arrays are printed with newlines between indented elements for readability.

-T [scalar[=value] array[=(value ...)] [sep]]

This flag has a different meaning when used with -f; see below. Oth? erwise the -T option requires zero, two, or three arguments to be present. With no arguments, the list of parameters created in this fashion is shown. With two or three arguments, the first two are the name of a scalar and of an array parameter (in that order) that will be tied together in the manner of $PATH and $path. The optional third argument is a single-character separator which will be used to join the elements of the array to form the scalar; if absent, a colon is used, as with $PATH. Only the first character of the separator is significant; any remaining characters are ignored. Multibyte charac? ters are not yet supported.

Only one of the scalar and array parameters may be assigned an ini? tial value (the restrictions on assignment forms described above also apply).

Both the scalar and the array may be manipulated as normal. If one is unset, the other will automatically be unset too. There is no way of untying the variables without unsetting them, nor of converting the type of one of them with another typeset command; +T does not work, assigning an array to scalar is an error, and assigning a scalar to array sets it to be a single-element array.

Note that both `typeset -xT ...' and `export -T ...' work, but only the scalar will be marked for export. Setting the value using the scalar version causes a split on all separators (which cannot be quoted). It is possible to apply -T to two previously tied variables

but with a different separator character, in which case the variables remain joined as before but the separator is changed.

When an existing scalar is tied to a new array, the value of the scalar is preserved but no attribute other than export will be pre? served.

Attribute flags that transform the final value (-L, -R, -Z, -l, -u) are only applied to the expanded value at the point of a parameter expansion expres? sion using `$'. They are not applied when a parameter is retrieved inter? nally by the shell for any purpose.

The following attribute flags may be specified:

-A The names refer to associative array parameters; see `Array Parame? ters' in zshparam(1).

-L [n]

Left justify and remove leading blanks from the value when the param? eter is expanded. If n is nonzero, it defines the width of the field. If n is zero, the width is determined by the width of the value of the first assignment. In the case of numeric parameters, the length of the complete value assigned to the parameter is used to determine the width, not the value that would be output.

The width is the count of characters, which may be multibyte charac? ters if the MULTIBYTE option is in effect. Note that the screen width of the character is not taken into account; if this is re? quired, use padding with parameter expansion flags ${(ml...)...} as described in `Parameter Expansion Flags' in zshexpn(1).

When the parameter is expanded, it is filled on the right with blanks or truncated if necessary to fit the field. Note truncation can lead to unexpected results with numeric parameters. Leading zeros are re? moved if the -Z flag is also set.

-R [n]

Similar to -L, except that right justification is used; when the pa? rameter is expanded, the field is left filled with blanks or trun? cated from the end. May not be combined with the -Z flag.

-U For arrays (but not for associative arrays), keep only the first oc?

currence of each duplicated value. This may also be set for tied pa? rameters (see -T) or colon-separated special parameters like PATH or FIGNORE, etc. Note the flag takes effect on assignment, and the type of the variable being assigned to is determinative; for variables with shared values it is therefore recommended to set the flag for all interfaces, e.g. `typeset -U PATH path'.

This flag has a different meaning when used with -f; see below.

-Z [n]

Specially handled if set along with the -L flag. Otherwise, similar to -R, except that leading zeros are used for padding instead of blanks if the first non-blank character is a digit. Numeric parame? ters are specially handled: they are always eligible for padding with zeroes, and the zeroes are inserted at an appropriate place in the output.

-a The names refer to array parameters. An array parameter may be cre? ated this way, but it may be assigned to in the typeset statement only if the reserved word form of typeset is enabled (as it is by de? fault). When displaying, both normal and associative arrays are shown.

-f The names refer to functions rather than parameters. No assignments can be made, and the only other valid flags are -t, -T, -k, -u, -U and -z. The flag -t turns on execution tracing for this function; the flag -T does the same, but turns off tracing for any named (not anonymous) function called from the present one, unless that function also has the -t or -T flag. The -u and -U flags cause the function to be marked for autoloading; -U also causes alias expansion to be suppressed when the function is loaded. See the description of the `autoload' builtin for details.

Note that the builtin functions provides the same basic capabilities as typeset -f but gives access to a few extra options; autoload gives further additional options for the case typeset -fu and typeset -fU.

-h Hide: only useful for special parameters (those marked `<S>' in the table in zshparam(1)), and for local parameters with the same name as

a special parameter, though harmless for others. A special parameter with this attribute will not retain its special effect when made lo? cal. Thus after `typeset -h PATH', a function containing `typeset PATH' will create an ordinary local parameter without the usual be? haviour of PATH. Alternatively, the local parameter may itself be given this attribute; hence inside a function `typeset -h PATH' cre? ates an ordinary local parameter and the special PATH parameter is not altered in any way. It is also possible to create a local param? eter using `typeset +h special', where the local copy of special will retain its special properties regardless of having the -h attribute. Global special parameters loaded from shell modules (currently those in zsh/mapfile and zsh/parameter) are automatically given the -h at? tribute to avoid name clashes.

-H Hide value: specifies that typeset will not display the value of the parameter when listing parameters; the display for such parameters is always as if the `+' flag had been given. Use of the parameter is in other respects normal, and the option does not apply if the parameter is specified by name, or by pattern with the -m option. This is on by default for the parameters in the zsh/parameter and zsh/mapfile modules. Note, however, that unlike the -h flag this is also useful for non-special parameters.

-i [n]

Use an internal integer representation. If n is nonzero it defines the output arithmetic base, otherwise it is determined by the first assignment. Bases from 2 to 36 inclusive are allowed.

-E [n]

Use an internal double-precision floating point representation. On output the variable will be converted to scientific notation. If n is nonzero it defines the number of significant figures to display; the default is ten.

-F [n]

Use an internal double-precision floating point representation. On output the variable will be converted to fixed-point decimal nota?

tion. If n is nonzero it defines the number of digits to display af? ter the decimal point; the default is ten.

-l Convert the result to lower case whenever the parameter is expanded. The value is not converted when assigned.

-r The given names are marked readonly. Note that if name is a special parameter, the readonly attribute can be turned on, but cannot then be turned off.

If the POSIX_BUILTINS option is set, the readonly attribute is more restrictive: unset variables can be marked readonly and cannot then be set; furthermore, the readonly attribute cannot be removed from any variable.

It is still possible to change other attributes of the variable though, some of which like -U or -Z would affect the value. More gen? erally, the readonly attribute should not be relied on as a security mechanism.

Note that in zsh (like in pdksh but unlike most other shells) it is still possible to create a local variable of the same name as this is considered a different variable (though this variable, too, can be marked readonly). Special variables that have been made readonly re? tain their value and readonly attribute when made local.

-t Tags the named parameters. Tags have no special meaning to the shell. This flag has a different meaning when used with -f; see above.

-u Convert the result to upper case whenever the parameter is expanded. The value is not converted when assigned. This flag has a different meaning when used with -f; see above.

-x Mark for automatic export to the environment of subsequently executed commands. If the option GLOBAL_EXPORT is set, this implies the op? tion -g, unless +g is also explicitly given; in other words the pa? rameter is not made local to the enclosing function. This is for compatibility with previous versions of zsh.

ulimit [-HSa] [{ -bcdfiklmnpqrsTtvwx | -N resource } [limit] ...]

Set or display resource limits of the shell and the processes started by the

shell. The value of limit can be a number in the unit specified below or one of the values `unlimited', which removes the limit on the resource, or `hard', which uses the current value of the hard limit on the resource.

By default, only soft limits are manipulated. If the -H flag is given use hard limits instead of soft limits. If the -S flag is given together with the -H flag set both hard and soft limits.

If no options are used, the file size limit (-f) is assumed.

If limit is omitted the current value of the specified resources are printed. When more than one resource value is printed, the limit name and unit is printed before each value.

When looping over multiple resources, the shell will abort immediately if it detects a badly formed argument. However, if it fails to set a limit for some other reason it will continue trying to set the remaining limits.

Not all the following resources are supported on all systems. Running ulimit -a will show which are supported.

-a Lists all of the current resource limits.

-b Socket buffer size in bytes (N.B. not kilobytes)

-c 512-byte blocks on the size of core dumps.

-d Kilobytes on the size of the data segment.

-f 512-byte blocks on the size of files written.

-i The number of pending signals.

-k The number of kqueues allocated.

-l Kilobytes on the size of locked-in memory.

-m Kilobytes on the size of physical memory.

-n open file descriptors.

-p The number of pseudo-terminals.

-q Bytes in POSIX message queues.

-r Maximum real time priority. On some systems where this is not avail? able, such as NetBSD, this has the same effect as -T for compatibil? ity with sh.

-s Kilobytes on the size of the stack.

-T The number of simultaneous threads available to the user.

-t CPU seconds to be used.

-u The number of processes available to the user.

-v Kilobytes on the size of virtual memory. On some systems this refers to the limit called `address space'.

-w Kilobytes on the size of swapped out memory.

-x The number of locks on files.

A resource may also be specified by integer in the form `-N resource', where resource corresponds to the integer defined for the resource by the operat? ing system. This may be used to set the limits for resources known to the shell which do not correspond to option letters. Such limits will be shown by number in the output of `ulimit -a'.

The number may alternatively be out of the range of limits compiled into the shell. The shell will try to read or write the limit anyway, and will re? port an error if this fails.

umask [-S] [mask]

The umask is set to mask. mask can be either an octal number or a symbolic value as described in chmod(1). If mask is omitted, the current value is printed. The -S option causes the mask to be printed as a symbolic value. Otherwise, the mask is printed as an octal number. Note that in the sym? bolic form the permissions you specify are those which are to be allowed (not denied) to the users specified.

unalias [-ams] name ...

Removes aliases. This command works the same as unhash -a, except that the -a option removes all regular or global aliases, or with -s all suffix aliases: in this case no name arguments may appear. The options -m (remove by pattern) and -s without -a (remove listed suffix aliases) behave as for unhash -a. Note that the meaning of -a is different between unalias and un? hash.

unfunction

Same as unhash -f.

unhash [-adfms] name ...

Remove the element named name from an internal hash table. The default is remove elements from the command hash table. The -a option causes unhash to remove regular or global aliases; note when removing a global aliases that

the argument must be quoted to prevent it from being expanded before being passed to the command. The -s option causes unhash to remove suffix aliases. The -f option causes unhash to remove shell functions. The -d op? tions causes unhash to remove named directories. If the -m flag is given the arguments are taken as patterns (should be quoted) and all elements of the corresponding hash table with matching names will be removed.

unlimit [-hs] resource ...

The resource limit for each resource is set to the hard limit. If the -h flag is given and the shell has appropriate privileges, the hard resource limit for each resource is removed. The resources of the shell process are only changed if the -s flag is given.

The unlimit command is not made available by default when the shell starts in a mode emulating another shell. It can be made available with the com? mand `zmodload -F zsh/rlimits b:unlimit'.

unset [-fmv] name ...

Each named parameter is unset. Local parameters remain local even if unset; they appear unset within scope, but the previous value will still reappear when the scope ends.

Individual elements of associative array parameters may be unset by using subscript syntax on name, which should be quoted (or the entire command pre? fixed with noglob) to protect the subscript from filename generation.

If the -m flag is specified the arguments are taken as patterns (should be quoted) and all parameters with matching names are unset. Note that this cannot be used when unsetting associative array elements, as the subscript will be treated as part of the pattern.

The -v flag specifies that name refers to parameters. This is the default behaviour.

unset -f is equivalent to unfunction.

unsetopt [{+|-}options | {+|-}o option_name] [name ...]

Unset the options for the shell. All options specified either with flags or by name are unset. If no arguments are supplied, the names of all options currently unset are printed. If the -m flag is given the arguments are taken as patterns (which should be quoted to preserve them from being inter?

preted as glob patterns), and all options with names matching these patterns are unset.

vared See the section `Zle Builtins' in zshzle(1).

wait [job ...]

Wait for the specified jobs or processes. If job is not given then all currently active child processes are waited for. Each job can be either a job specification or the process ID of a job in the job table. The exit status from this command is that of the job waited for. If job represents an unknown job or process ID, a warning is printed (unless the POSIX_BUILTINS option is set) and the exit status is 127.

It is possible to wait for recent processes (specified by process ID, not by job) that were running in the background even if the process has exited. Typically the process ID will be recorded by capturing the value of the variable $! immediately after the process has been started. There is a limit on the number of process IDs remembered by the shell; this is given by the value of the system configuration parameter CHILD_MAX. When this limit is reached, older process IDs are discarded, least recently started processes first.

Note there is no protection against the process ID wrapping, i.e. if the wait is not executed soon enough there is a chance the process waited for is the wrong one. A conflict implies both process IDs have been generated by the shell, as other processes are not recorded, and that the user is potentially interested in both, so this problem is intrinsic to process IDs.

whence [-vcwfpamsS] [-x num] name ...

For each name, indicate how it would be interpreted if used as a command name.

If name is not an alias, built-in command, external command, shell function, hashed command, or a reserved word, the exit status shall be non-zero, and -- if -v, -c, or -w was passed -- a message will be written to standard output. (This is different from other shells that write that message to standard error.)

whence is most useful when name is only the last path component of a command, i.e. does not include a `/'; in particular, pattern matching only suc?

ceeds if just the non-directory component of the command is passed.

-v Produce a more verbose report.

-c Print the results in a csh-like format. This takes precedence over -v.

-w For each name, print `name: word' where word is one of alias, builtin, command, function, hashed, reserved or none, according as name corresponds to an alias, a built-in command, an external com? mand, a shell function, a command defined with the hash builtin, a reserved word, or is not recognised. This takes precedence over -v and -c.

-f Causes the contents of a shell function to be displayed, which would otherwise not happen unless the -c flag were used.

-p Do a path search for name even if it is an alias, reserved word, shell function or builtin.

-a Do a search for all occurrences of name throughout the command path. Normally only the first occurrence is printed.

-m The arguments are taken as patterns (pattern characters should be quoted), and the information is displayed for each command matching one of these patterns.

-s If a pathname contains symlinks, print the symlink-free pathname as well.

-S As -s, but if the pathname had to be resolved by following multiple symlinks, the intermediate steps are printed, too. The symlink re? solved at each step might be anywhere in the path.

-x num Expand tabs when outputting shell functions using the -c option. This has the same effect as the -x option to the functions builtin.

where [-wpmsS] [-x num] name ...

Equivalent to whence -ca.

which [-wpamsS] [-x num] name ...

Equivalent to whence -c.

zcompile [-U] [-z | -k] [-R | -M] file [name ...]

zcompile -ca [-m] [-R | -M] file [name ...]

zcompile -t file [name ...]

This builtin command can be used to compile functions or scripts, storing the compiled form in a file, and to examine files containing the compiled form. This allows faster autoloading of functions and sourcing of scripts by avoiding parsing of the text when the files are read.

The first form (without the -c, -a or -t options) creates a compiled file. If only the file argument is given, the output file has the name `file.zwc' and will be placed in the same directory as the file. The shell will load the compiled file instead of the normal function file when the function is autoloaded; see the section `Autoloading Functions' in zshmisc(1) for a de? scription of how autoloaded functions are searched. The extension .zwc stands for `zsh word code'.

If there is at least one name argument, all the named files are compiled into the output file given as the first argument. If file does not end in .zwc, this extension is automatically appended. Files containing multiple compiled functions are called `digest' files, and are intended to be used as elements of the FPATH/fpath special array.

The second form, with the -c or -a options, writes the compiled definitions for all the named functions into file. For -c, the names must be functions currently defined in the shell, not those marked for autoloading. Undefined functions that are marked for autoloading may be written by using the -a op? tion, in which case the fpath is searched and the contents of the definition files for those functions, if found, are compiled into file. If both -c and -a are given, names of both defined functions and functions marked for au? toloading may be given. In either case, the functions in files written with the -c or -a option will be autoloaded as if the KSH_AUTOLOAD option were unset.

The reason for handling loaded and not-yet-loaded functions with different options is that some definition files for autoloading define multiple func? tions, including the function with the same name as the file, and, at the end, call that function. In such cases the output of `zcompile -c' does not include the additional functions defined in the file, and any other initial? ization code in the file is lost. Using `zcompile -a' captures all this ex? tra information.

If the -m option is combined with -c or -a, the names are used as patterns and all functions whose names match one of these patterns will be written. If no name is given, the definitions of all functions currently defined or marked as autoloaded will be written.

Note the second form cannot be used for compiling functions that include redirections as part of the definition rather than within the body of the function; for example

```
fn1() { { ... } >~/logfile }
```

can be compiled but

```
fn1() { ... } >~/logfile
```

cannot. It is possible to use the first form of zcompile to compile au? toloadable functions that include the full function definition instead of just the body of the function.

The third form, with the -t option, examines an existing compiled file. Without further arguments, the names of the original files compiled into it are listed. The first line of output shows the version of the shell which compiled the file and how the file will be used (i.e. by reading it directly or by mapping it into memory). With arguments, nothing is output and the return status is set to zero if definitions for all names were found in the compiled file, and non-zero if the definition for at least one name was not found.

Other options:

-U Aliases are not expanded when compiling the named files.

-R When the compiled file is read, its contents are copied into the shell's memory, rather than memory-mapped (see -M). This happens au? tomatically on systems that do not support memory mapping.
When compiling scripts instead of autoloadable functions, it is often desirable to use this option; otherwise the whole file, including the code to define functions which have already been defined, will remain mapped, consequently wasting memory.

-M The compiled file is mapped into the shell's memory when read. This is done in such a way that multiple instances of the shell running on the same host will share this mapped file. If neither -R nor -M is

given, the zcompile builtin decides what to do based on the size of the compiled file.

-k

-z These options are used when the compiled file contains functions which are to be autoloaded. If -z is given, the function will be au? toloaded as if the KSH_AUTOLOAD option is not set, even if it is set at the time the compiled file is read, while if the -k is given, the function will be loaded as if KSH_AUTOLOAD is set. These options also take precedence over any -k or -z options specified to the au? toload builtin. If neither of these options is given, the function will be loaded as determined by the setting of the KSH_AUTOLOAD op? tion at the time the compiled file is read.

These options may also appear as many times as necessary between the listed names to specify the loading style of all following functions, up to the next -k or -z.

The created file always contains two versions of the compiled format, one for big-endian machines and one for small-endian machines. The upshot of this is that the compiled file is machine independent and if it is read or mapped, only one half of the file is actually used (and mapped).

zformat

See the section `The zsh/zutil Module' in zshmodules(1).

zftp See the section `The zsh/zftp Module' in zshmodules(1).

zle See the section `Zle Builtins' in zshzle(1).

zmodload [-dL] [-s] [...]

zmodload -F [-alLme -P param] module [[+-]feature ...]

zmodload -e [-A] [...]

zmodload [-a [-bcpf [-I]]] [-iL] ...

zmodload -u [-abcdpf [-I]] [-iL] ...

zmodload -A [-L] [modalias[=module] ...]

zmodload -R modalias ...

Performs operations relating to zsh's loadable modules. Loading of modules while the shell is running (`dynamical loading') is not available on all op?

erating systems, or on all installations on a particular operating system, although the zmodload command itself is always available and can be used to manipulate modules built into versions of the shell executable without dy? namical loading.

Without arguments the names of all currently loaded binary modules are printed. The -L option causes this list to be in the form of a series of zmodload commands. Forms with arguments are:

zmodload [-is] name ...

zmodload -u [-i] name ...

In the simplest case, zmodload loads a binary module. The module must be in a file with a name consisting of the specified name fol? lowed by a standard suffix, usually `.so' (`.sl' on HPUX). If the module to be loaded is already loaded the duplicate module is ig? nored. If zmodload detects an inconsistency, such as an invalid mod? ule name or circular dependency list, the current code block is aborted. If it is available, the module is loaded if necessary, while if it is not available, non-zero status is silently returned.

The option -i is accepted for compatibility but has no effect.

The named module is searched for in the same way a command is, using $module_path instead of $path. However, the path search is performed even when the module name contains a `/', which it usually does. There is no way to prevent the path search.

If the module supports features (see below), zmodload tries to enable all features when loading a module. If the module was successfully loaded but not all features could be enabled, zmodload returns status 2.

If the option -s is given, no error is printed if the module was not available (though other errors indicating a problem with the module are printed). The return status indicates if the module was loaded. This is appropriate if the caller considers the module optional.

With -u, zmodload unloads modules. The same name must be given that was given when the module was loaded, but it is not necessary for the module to exist in the file system. The -i option suppresses the er?

ror if the module is already unloaded (or was never loaded).

Each module has a boot and a cleanup function. The module will not be loaded if its boot function fails. Similarly a module can only be unloaded if its cleanup function runs successfully.

zmodload -F [-almLe -P param] module [[+-]feature ...]

zmodload -F allows more selective control over the features provided by modules. With no options apart from -F, the module named module is loaded, if it was not already loaded, and the list of features is set to the required state. If no features are specified, the module is loaded, if it was not already loaded, but the state of features is unchanged. Each feature may be preceded by a + to turn the feature on, or - to turn it off; the + is assumed if neither character is present. Any feature not explicitly mentioned is left in its current state; if the module was not previously loaded this means any such features will remain disabled. The return status is zero if all fea? tures were set, 1 if the module failed to load, and 2 if some fea? tures could not be set (for example, a parameter couldn't be added because there was a different parameter of the same name) but the module was loaded.

The standard features are builtins, conditions, parameters and math functions; these are indicated by the prefix `b:', `c:' (`C:' for an infix condition), `p:' and `f:', respectively, followed by the name that the corresponding feature would have in the shell. For example, `b:strftime' indicates a builtin named strftime and p:EPOCHSECONDS indicates a parameter named EPOCHSECONDS. The module may provide other (`abstract') features of its own as indicated by its documenta? tion; these have no prefix.

With -l or -L, features provided by the module are listed. With -l alone, a list of features together with their states is shown, one feature per line. With -L alone, a zmodload -F command that would cause enabled features of the module to be turned on is shown. With -lL, a zmodload -F command that would cause all the features to be set to their current state is shown. If one of these combinations is

given with the option -P param then the parameter param is set to an array of features, either features together with their state or (if -L alone is given) enabled features.

With the option -L the module name may be omitted; then a list of all enabled features for all modules providing features is printed in the form of zmodload -F commands. If -l is also given, the state of both enabled and disabled features is output in that form.

A set of features may be provided together with -l or -L and a module name; in that case only the state of those features is considered. Each feature may be preceded by + or - but the character has no ef? fect. If no set of features is provided, all features are consid? ered.

With -e, the command first tests that the module is loaded; if it is not, status 1 is returned. If the module is loaded, the list of fea? tures given as an argument is examined. Any feature given with no prefix is simply tested to see if the module provides it; any feature given with a prefix + or - is tested to see if is provided and in the given state. If the tests on all features in the list succeed, sta? tus 0 is returned, else status 1.

With -m, each entry in the given list of features is taken as a pat? tern to be matched against the list of features provided by the mod? ule. An initial + or - must be given explicitly. This may not be combined with the -a option as autoloads must be specified explic? itly.

With -a, the given list of features is marked for autoload from the specified module, which may not yet be loaded. An optional + may ap? pear before the feature name. If the feature is prefixed with -, any existing autoload is removed. The options -l and -L may be used to list autoloads. Autoloading is specific to individual features; when the module is loaded only the requested feature is enabled. Autoload requests are preserved if the module is subsequently unloaded until an explicit `zmodload -Fa module -feature' is issued. It is not an error to request an autoload for a feature of a module that is al?

ready loaded.

When the module is loaded each autoload is checked against the fea? tures actually provided by the module; if the feature is not provided the autoload request is deleted. A warning message is output; if the module is being loaded to provide a different feature, and that au? toload is successful, there is no effect on the status of the current command. If the module is already loaded at the time when zmodload -Fa is run, an error message is printed and status 1 returned.

zmodload -Fa can be used with the -l, -L, -e and -P options for list? ing and testing the existence of autoloadable features. In this case -l is ignored if -L is specified. zmodload -FaL with no module name lists autoloads for all modules.

Note that only standard features as described above can be au? toloaded; other features require the module to be loaded before en? abling.

zmodload -d [-L] [name]

zmodload -d name dep ...

zmodload -ud name [dep ...]

The -d option can be used to specify module dependencies. The mod? ules named in the second and subsequent arguments will be loaded be? fore the module named in the first argument.

With -d and one argument, all dependencies for that module are listed. With -d and no arguments, all module dependencies are listed. This listing is by default in a Makefile-like format. The -L option changes this format to a list of zmodload -d commands.

If -d and -u are both used, dependencies are removed. If only one argument is given, all dependencies for that module are removed.

zmodload -ab [-L]

zmodload -ab [-i] name [builtin ...]

zmodload -ub [-i] builtin ...

The -ab option defines autoloaded builtins. It defines the specified builtins. When any of those builtins is called, the module specified in the first argument is loaded and all its features are enabled (for

selective control of features use `zmodload -F -a' as described above). If only the name is given, one builtin is defined, with the same name as the module. -i suppresses the error if the builtin is already defined or autoloaded, but not if another builtin of the same name is already defined.

With -ab and no arguments, all autoloaded builtins are listed, with the module name (if different) shown in parentheses after the builtin name. The -L option changes this format to a list of zmodload -a commands.

If -b is used together with the -u option, it removes builtins previ? ously defined with -ab. This is only possible if the builtin is not yet loaded. -i suppresses the error if the builtin is already re? moved (or never existed).

Autoload requests are retained if the module is subsequently unloaded until an explicit `zmodload -ub builtin' is issued.

zmodload -ac [-IL]

zmodload -ac [-iI] name [cond ...]

zmodload -uc [-iI] cond ...

The -ac option is used to define autoloaded condition codes. The cond strings give the names of the conditions defined by the module. The optional -I option is used to define infix condition names. Without this option prefix condition names are defined.

If given no condition names, all defined names are listed (as a se? ries of zmodload commands if the -L option is given).

The -uc option removes definitions for autoloaded conditions.

zmodload -ap [-L]

zmodload -ap [-i] name [parameter ...]

zmodload -up [-i] parameter ...

The -p option is like the -b and -c options, but makes zmodload work on autoloaded parameters instead.

zmodload -af [-L]

zmodload -af [-i] name [function ...]

zmodload -uf [-i] function ...

The -f option is like the -b, -p, and -c options, but makes zmodload work on autoloaded math functions instead.

zmodload -a [-L]

zmodload -a [-i] name [builtin ...]

zmodload -ua [-i] builtin ...

Equivalent to -ab and -ub.

zmodload -e [-A] [string ...]

The -e option without arguments lists all loaded modules; if the -A option is also given, module aliases corresponding to loaded modules are also shown. If arguments are provided, nothing is printed; the return status is set to zero if all strings given as arguments are names of loaded modules and to one if at least on string is not the name of a loaded module. This can be used to test for the availabil? ity of things implemented by modules. In this case, any aliases are automatically resolved and the -A flag is not used.

zmodload -A [-L] [modalias[=module] ...]

For each argument, if both modalias and module are given, define modalias to be an alias for the module module. If the module modalias is ever subsequently requested, either via a call to zmod? load or implicitly, the shell will attempt to load module instead.

If module is not given, show the definition of modalias. If no argu? ments are given, list all defined module aliases. When listing, if the -L flag was also given, list the definition as a zmodload command to recreate the alias.

The existence of aliases for modules is completely independent of whether the name resolved is actually loaded as a module: while the alias exists, loading and unloading the module under any alias has exactly the same effect as using the resolved name, and does not af? fect the connection between the alias and the resolved name which can be removed either by zmodload -R or by redefining the alias. Chains of aliases (i.e. where the first resolved name is itself an alias) are valid so long as these are not circular. As the aliases take the same format as module names, they may include path separators: in

this case, there is no requirement for any part of the path named to exist as the alias will be resolved first. For example, `any/old/alias' is always a valid alias.

Dependencies added to aliased modules are actually added to the re? solved module; these remain if the alias is removed. It is valid to create an alias whose name is one of the standard shell modules and which resolves to a different module. However, if a module has de? pendencies, it will not be possible to use the module name as an alias as the module will already be marked as a loadable module in its own right.

Apart from the above, aliases can be used in the zmodload command anywhere module names are required. However, aliases will not be shown in lists of loaded modules with a bare `zmodload'.

zmodload -R modalias ...

For each modalias argument that was previously defined as a module alias via zmodload -A, delete the alias. If any was not defined, an error is caused and the remainder of the line is ignored.

Note that zsh makes no distinction between modules that were linked into the shell and modules that are loaded dynamically. In both cases this builtin command has to be used to make available the builtins and other things de? fined by modules (unless the module is autoloaded on these definitions). This is true even for systems that don't support dynamic loading of modules.

zparseopts

See the section `The zsh/zutil Module' in zshmodules(1).

zprof See the section `The zsh/zprof Module' in zshmodules(1).

zpty See the section `The zsh/zpty Module' in zshmodules(1).

zregexparse

See the section `The zsh/zutil Module' in zshmodules(1).

zsocket

See the section `The zsh/net/socket Module' in zshmodules(1).

zstyle See the section `The zsh/zutil Module' in zshmodules(1).

ztcp See the section `The zsh/net/tcp Module' in zshmodules(1).