
***Rocky Enterprise Linux 9.2 Manual Pages on command 'zshoptions.1'***

***$ man zshoptions.1***

ZSHOPTIONS(1) General Commands Manual ZSHOPTIONS(1)

NAME

zshoptions - zsh options

SPECIFYING OPTIONS

Options are primarily referred to by name. These names are case insen?
sitive and underscores are ignored. For example, `allexport' is equiv?
alent to `A__lleXP_ort'.

The sense of an option name may be inverted by preceding it with `no',
so `setopt No_Beep' is equivalent to `unsetopt beep'. This inversion
can only be done once, so `nonobeep' is not a synonym for `beep'. Sim?
ilarly, `tify' is not a synonym for `nonotify' (the inversion of `no?
tify').

Some options also have one or more single letter names. There are two
sets of single letter options: one used by default, and another used to
emulate sh/ksh (used when the SH_OPTION_LETTERS option is set). The
single letter options can be used on the shell command line, or with
the set, setopt and unsetopt builtins, as normal Unix options preceded
by `-'.

The sense of the single letter options may be inverted by using `+' in? stead of `-'. Some of the single letter option names refer to an op? tion being off, in which case the inversion of that name refers to the option being on. For example, `+n' is the short name of `exec', and `-n' is the short name of its inversion, `noexec'.

In strings of single letter options supplied to the shell at startup, trailing whitespace will be ignored; for example the string `-f ' will be treated just as `-f', but the string `-f i' is an error. This is because many systems which implement the `#!' mechanism for calling scripts do not strip trailing whitespace.

## DESCRIPTION OF OPTIONS

In the following list, options set by default in all emulations are marked <D>; those set by default only in csh, ksh, sh, or zsh emula? tions are marked <C>, <K>, <S>, <Z> as appropriate. When listing op? tions (by `setopt', `unsetopt', `set -o' or `set +o'), those turned on by default appear in the list prefixed with `no'. Hence (unless KSH_OPTION_PRINT is set), `setopt' shows all options whose settings are changed from the default.

### Changing Directories

AUTO_CD (-J)

If a command is issued that can't be executed as a normal com? mand, and the command is the name of a directory, perform the cd command to that directory. This option is only applicable if the option SHIN_STDIN is set, i.e. if commands are being read from standard input. The option is designed for interactive use; it is recommended that cd be used explicitly in scripts to avoid ambiguity.

AUTO_PUSHD (-N)

Make cd push the old directory onto the directory stack.

CDABLE_VARS (-T)

If the argument to a cd command (or an implied cd with the AUTO_CD option set) is not a directory, and does not begin with a slash, try to expand the expression as if it were preceded by

a `~' (see the section `Filename Expansion').

CD_SILENT

Never print the working directory after a cd (whether explicit or implied with the AUTO_CD option set). cd normally prints the working directory when the argument given to it was -, a stack entry, or the name of a directory found under CDPATH. Note that this is distinct from pushd's stack-printing behaviour, which is controlled by PUSHD_SILENT. This option overrides the print?ing-related effects of POSIX_CD.

CHASE_DOTS

When changing to a directory containing a path segment `..' which would otherwise be treated as canceling the previous seg?ment in the path (in other words, `foo/..' would be removed from the path, or if `..' is the first part of the path, the last part of the current working directory would be removed), instead resolve the path to the physical directory. This option is overridden by CHASE_LINKS.

For example, suppose /foo/bar is a link to the directory /alt/rod. Without this option set, `cd /foo/bar/..' changes to /foo; with it set, it changes to /alt. The same applies if the current directory is /foo/bar and `cd ..' is used. Note that all other symbolic links in the path will also be resolved.

CHASE_LINKS (-w)

Resolve symbolic links to their true values when changing direc?tory. This also has the effect of CHASE_DOTS, i.e. a `..' path segment will be treated as referring to the physical parent, even if the preceding path segment is a symbolic link.

POSIX_CD <K> <S>

Modifies the behaviour of cd, chdir and pushd commands to make them more compatible with the POSIX standard. The behaviour with the option unset is described in the documentation for the cd builtin in zshbuiltins(1). If the option is set, the shell does not test for directories beneath the local directory (`.') until

after all directories in cdpath have been tested, and the cd and chdir commands do not recognise arguments of the form `{+|-}n' as directory stack entries.

Also, if the option is set, the conditions under which the shell prints the new directory after changing to it are modified. It is no longer restricted to interactive shells (although printing of the directory stack with pushd is still limited to interac? tive shells); and any use of a component of CDPATH, including a `.' but excluding an empty component that is otherwise treated as `.', causes the directory to be printed.

PUSHD_IGNORE_DUPS

Don't push multiple copies of the same directory onto the direc? tory stack.

PUSHD_MINUS

Exchanges the meanings of `+' and `-' when used with a number to specify a directory in the stack.

PUSHD_SILENT (-E)

Do not print the directory stack after pushd or popd.

PUSHD_TO_HOME (-D)

Have pushd with no arguments act like `pushd $HOME'.

Completion

ALWAYS_LAST_PROMPT <D>

If unset, key functions that list completions try to return to the last prompt if given a numeric argument. If set these func? tions try to return to the last prompt if given no numeric argu? ment.

ALWAYS_TO_END

If a completion is performed with the cursor within a word, and a full completion is inserted, the cursor is moved to the end of the word. That is, the cursor is moved to the end of the word if either a single match is inserted or menu completion is per? formed.

AUTO_LIST (-9) <D>

Automatically list choices on an ambiguous completion.

AUTO_MENU <D>

Automatically use menu completion after the second consecutive request for completion, for example by pressing the tab key repeatedly. This option is overridden by MENU_COMPLETE.

AUTO_NAME_DIRS

Any parameter that is set to the absolute name of a directory immediately becomes a name for that directory, that will be used by the `%~' and related prompt sequences, and will be available when completion is performed on a word starting with `~'. (Otherwise, the parameter must be used in the form `~param' first.)

AUTO_PARAM_KEYS <D>

If a parameter name was completed and a following character (normally a space) automatically inserted, and the next character typed is one of those that have to come directly after the name (like `}', `:', etc.), the automatically added character is deleted, so that the character typed comes immediately after the parameter name. Completion in a brace expansion is affected similarly: the added character is a `,', which will be removed if `}' is typed next.

AUTO_PARAM_SLASH <D>

If a parameter is completed whose content is the name of a directory, then add a trailing slash instead of a space.

AUTO_REMOVE_SLASH <D>

When the last character resulting from a completion is a slash and the next character typed is a word delimiter, a slash, or a character that ends a command (such as a semicolon or an ampersand), remove the slash.

BASH_AUTO_LIST

On an ambiguous completion, automatically list choices when the completion function is called twice in succession. This takes precedence over AUTO_LIST. The setting of LIST_AMBIGUOUS is respected. If AUTO_MENU is set, the menu behaviour will then

start with the third press. Note that this will not work with MENU_COMPLETE, since repeated completion calls immediately cycle through the list in that case.

COMPLETE_ALIASES

Prevents aliases on the command line from being internally substituted before completion is attempted. The effect is to make the alias a distinct command for completion purposes.

COMPLETE_IN_WORD

If unset, the cursor is set to the end of the word if completion is started. Otherwise it stays there and completion is done from both ends.

GLOB_COMPLETE

When the current word has a glob pattern, do not insert all the words resulting from the expansion but generate matches as for completion and cycle through them like MENU_COMPLETE. The matches are generated as if a `*' was added to the end of the word, or inserted at the cursor when COMPLETE_IN_WORD is set. This actually uses pattern matching, not globbing, so it works not only for files but for any completion, such as options, user names, etc.

Note that when the pattern matcher is used, matching control (for example, case-insensitive or anchored matching) cannot be used. This limitation only applies when the current word contains a pattern; simply turning on the GLOB_COMPLETE option does not have this effect.

HASH_LIST_ALL <D>

Whenever a command completion or spelling correction is attempted, make sure the entire command path is hashed first. This makes the first completion slower but avoids false reports of spelling errors.

LIST_AMBIGUOUS <D>

This option works when AUTO_LIST or BASH_AUTO_LIST is also set. If there is an unambiguous prefix to insert on the command line,

that is done without a completion list being displayed; in other words, auto-listing behaviour only takes place when nothing would be inserted. In the case of BASH_AUTO_LIST, this means that the list will be delayed to the third call of the function.

LIST_BEEP <D>

Beep on an ambiguous completion. More accurately, this forces the completion widgets to return status 1 on an ambiguous com? pletion, which causes the shell to beep if the option BEEP is also set; this may be modified if completion is called from a user-defined widget.

LIST_PACKED

Try to make the completion list smaller (occupying less lines) by printing the matches in columns with different widths.

LIST_ROWS_FIRST

Lay out the matches in completion lists sorted horizontally, that is, the second match is to the right of the first one, not under it as usual.

LIST_TYPES (-X) <D>

When listing files that are possible completions, show the type of each file with a trailing identifying mark.

MENU_COMPLETE (-Y)

On an ambiguous completion, instead of listing possibilities or beeping, insert the first match immediately. Then when comple? tion is requested again, remove the first match and insert the second match, etc. When there are no more matches, go back to the first one again. reverse-menu-complete may be used to loop through the list in the other direction. This option overrides AUTO_MENU.

REC_EXACT (-S)

If the string on the command line exactly matches one of the possible completions, it is accepted, even if there is another completion (i.e. that string with something else added) that also matches.

Expansion and Globbing

BAD_PATTERN (+2) <C> <Z>

If a pattern for filename generation is badly formed, print an error message. (If this option is unset, the pattern will be left unchanged.)

BARE_GLOB_QUAL <Z>

In a glob pattern, treat a trailing set of parentheses as a qualifier list, if it contains no `|', `(' or (if special) `~' characters. See the section `Filename Generation'.

BRACE_CCL

Expand expressions in braces which would not otherwise undergo brace expansion to a lexically ordered list of all the charac? ters. See the section `Brace Expansion'.

CASE_GLOB <D>

Make globbing (filename generation) sensitive to case. Note that other uses of patterns are always sensitive to case. If the option is unset, the presence of any character which is spe? cial to filename generation will cause case-insensitive match? ing. For example, cvs(/) can match the directory CVS owing to the presence of the globbing flag (unless the option BARE_GLOB_QUAL is unset).

CASE_MATCH <D>

Make regular expressions using the zsh/regex module (including matches with =~) sensitive to case.

CSH_NULL_GLOB <C>

If a pattern for filename generation has no matches, delete the pattern from the argument list; do not report an error unless all the patterns in a command have no matches. Overrides NO? MATCH.

EQUALS <Z>

Perform = filename expansion. (See the section `Filename Expan? sion'.)

EXTENDED_GLOB

Treat the `#', `~' and `^' characters as part of patterns for filename generation, etc. (An initial unquoted `~' always pro? duces named directory expansion.)

FORCE_FLOAT

Constants in arithmetic evaluation will be treated as floating point even without the use of a decimal point; the values of in? teger variables will be converted to floating point when used in arithmetic expressions. Integers in any base will be converted.

GLOB (+F, ksh: +f) <D>

Perform filename generation (globbing). (See the section `File? name Generation'.)

GLOB_ASSIGN <C>

If this option is set, filename generation (globbing) is per? formed on the right hand side of scalar parameter assignments of the form `name=pattern (e.g. `foo=*'). If the result has more than one word the parameter will become an array with those words as arguments. This option is provided for backwards com? patibility only: globbing is always performed on the right hand side of array assignments of the form `name=(value)' (e.g. `foo=(*)') and this form is recommended for clarity; with this option set, it is not possible to predict whether the result will be an array or a scalar.

GLOB_DOTS (-4)

Do not require a leading `.' in a filename to be matched explic? itly.

GLOB_STAR_SHORT

When this option is set and the default zsh-style globbing is in effect, the pattern `**/*' can be abbreviated to `**' and the pattern `***/*' can be abbreviated to ***. Hence `**.c' finds a file ending in .c in any subdirectory, and `***.c' does the same while also following symbolic links. A / immediately after the `**' or `***' forces the pattern to be treated as the unabbrevi? ated form.

GLOB_SUBST <C> <K> <S>

Treat any characters resulting from parameter expansion as being eligible for filename expansion and filename generation, and any characters resulting from command substitution as being eligible for filename generation. Braces (and commas in between) do not become eligible for expansion.

HIST_SUBST_PATTERN

Substitutions using the :s and :& history modifiers are per? formed with pattern matching instead of string matching. This occurs wherever history modifiers are valid, including glob qualifiers and parameters. See the section Modifiers in zsh? expn(1).

IGNORE_BRACES (-I) <S>

Do not perform brace expansion. For historical reasons this also includes the effect of the IGNORE_CLOSE_BRACES option.

IGNORE_CLOSE_BRACES

When neither this option nor IGNORE_BRACES is set, a sole close brace character `}' is syntactically significant at any point on a command line. This has the effect that no semicolon or new? line is necessary before the brace terminating a function or current shell construct. When either option is set, a closing brace is syntactically significant only in command position. Unlike IGNORE_BRACES, this option does not disable brace expan? sion.

For example, with both options unset a function may be defined in the following fashion:

```
args() { echo $# }
```

while if either option is set, this does not work and something equivalent to the following is required:

```
args() { echo $#; }
```

KSH_GLOB <K>

In pattern matching, the interpretation of parentheses is af? fected by a preceding `@', `*', `+', `?' or `!'. See the sec?

tion `Filename Generation'.

MAGIC_EQUAL_SUBST

All unquoted arguments of the form `anything=expression' appear? ing after the command name have filename expansion (that is, where expression has a leading `~' or `=') performed on expres? sion as if it were a parameter assignment. The argument is not otherwise treated specially; it is passed to the command as a single argument, and not used as an actual parameter assignment. For example, in echo foo=~/bar:~/rod, both occurrences of ~ would be replaced. Note that this happens anyway with typeset and similar statements.

This option respects the setting of the KSH_TYPESET option. In other words, if both options are in effect, arguments looking like assignments will not undergo word splitting.

MARK_DIRS (-8, ksh: -X)

Append a trailing `/' to all directory names resulting from filename generation (globbing).

MULTIBYTE <D>

Respect multibyte characters when found in strings. When this option is set, strings are examined using the system library to determine how many bytes form a character, depending on the cur? rent locale. This affects the way characters are counted in pattern matching, parameter values and various delimiters.

The option is on by default if the shell was compiled with MULTIBYTE_SUPPORT; otherwise it is off by default and has no ef? fect if turned on.

If the option is off a single byte is always treated as a single character. This setting is designed purely for examining strings known to contain raw bytes or other values that may not be characters in the current locale. It is not necessary to un? set the option merely because the character set for the current locale does not contain multibyte characters.

The option does not affect the shell's editor, which always

uses the locale to determine multibyte characters. This is be? cause the character set displayed by the terminal emulator is independent of shell settings.

NOMATCH (+3) <C> <Z>

If a pattern for filename generation has no matches, print an error, instead of leaving it unchanged in the argument list. This also applies to file expansion of an initial `~' or `='.

NULL_GLOB (-G)

If a pattern for filename generation has no matches, delete the pattern from the argument list instead of reporting an error. Overrides NOMATCH.

NUMERIC_GLOB_SORT

If numeric filenames are matched by a filename generation pat? tern, sort the filenames numerically rather than lexicographi? cally.

RC_EXPAND_PARAM (-P)

Array expansions of the form `foo${xx}bar', where the parameter xx is set to (a b c), are substituted with `fooabar foobbar foocbar' instead of the default `fooa b cbar'. Note that an empty array will therefore cause all arguments to be removed.

REMATCH_PCRE

If set, regular expression matching with the =~ operator will use Perl-Compatible Regular Expressions from the PCRE library. (The zsh/pcre module must be available.) If not set, regular expressions will use the extended regexp syntax provided by the system libraries.

SH_GLOB <K> <S>

Disables the special meaning of `(', `|', `)' and '<' for glob? bing the result of parameter and command substitutions, and in some other places where the shell accepts patterns. If SH_GLOB is set but KSH_GLOB is not, the shell allows the interpretation of subshell expressions enclosed in parentheses in some cases where there is no space before the opening parenthesis, e.g.

!(true) is interpreted as if there were a space after the !.

This option is set by default if zsh is invoked as sh or ksh.

UNSET (+u, ksh: +u) <K> <S> <Z>

Treat unset parameters as if they were empty when substituting, and as if they were zero when reading their values in arithmetic expansion and arithmetic commands. Otherwise they are treated as an error.

WARN_CREATE_GLOBAL

Print a warning message when a global parameter is created in a function by an assignment or in math context. This often indi? cates that a parameter has not been declared local when it should have been. Parameters explicitly declared global from within a function using typeset -g do not cause a warning. Note that there is no warning when a local parameter is assigned to in a nested function, which may also indicate an error.

WARN_NESTED_VAR

Print a warning message when an existing parameter from an en? closing function scope, or global, is set in a function by an assignment or in math context. Assignment to shell special pa? rameters does not cause a warning. This is the companion to WARN_CREATE_GLOBAL as in this case the warning is only printed when a parameter is not created. Where possible, use of typeset -g to set the parameter suppresses the error, but note that this needs to be used every time the parameter is set. To restrict the effect of this option to a single function scope, use `func? tions -W'.

For example, the following code produces a warning for the as? signment inside the function nested as that overrides the value within toplevel

```
toplevel() {
  local foo="in fn"
  nested
}
```

```
nested() {
    foo="in nested"
}
setopt warn_nested_var
toplevel
```

History

APPEND_HISTORY <D>

If this is set, zsh sessions will append their history list to the history file, rather than replace it. Thus, multiple paral? lel zsh sessions will all have the new entries from their his? tory lists added to the history file, in the order that they exit. The file will still be periodically re-written to trim it when the number of lines grows 20% beyond the value specified by $SAVEHIST (see also the HIST_SAVE_BY_COPY option).

BANG_HIST (+K) <C> <Z>

Perform textual history expansion, csh-style, treating the char? acter `!' specially.

EXTENDED_HISTORY <C>

Save each command's beginning timestamp (in seconds since the epoch) and the duration (in seconds) to the history file. The format of this prefixed data is:
`: <beginning time>:<elapsed seconds>;<command>'.

HIST_ALLOW_CLOBBER

Add `|' to output redirections in the history. This allows his? tory references to clobber files even when CLOBBER is unset.

HIST_BEEP <D>

Beep in ZLE when a widget attempts to access a history entry which isn't there.

HIST_EXPIRE_DUPS_FIRST

If the internal history needs to be trimmed to add the current command line, setting this option will cause the oldest history event that has a duplicate to be lost before losing a unique event from the list. You should be sure to set the value of

HISTSIZE to a larger number than SAVEHIST in order to give you some room for the duplicated events, otherwise this option will behave just like HIST_IGNORE_ALL_DUPS once the history fills up with unique events.

HIST_FCNTL_LOCK

When writing out the history file, by default zsh uses ad-hoc file locking to avoid known problems with locking on some oper? ating systems. With this option locking is done by means of the system's fcntl call, where this method is available. On recent operating systems this may provide better performance, in par? ticular avoiding history corruption when files are stored on NFS.

HIST_FIND_NO_DUPS

When searching for history entries in the line editor, do not display duplicates of a line previously found, even if the du? plicates are not contiguous.

HIST_IGNORE_ALL_DUPS

If a new command line being added to the history list duplicates an older one, the older command is removed from the list (even if it is not the previous event).

HIST_IGNORE_DUPS (-h)

Do not enter command lines into the history list if they are du? plicates of the previous event.

HIST_IGNORE_SPACE (-g)

Remove command lines from the history list when the first char? acter on the line is a space, or when one of the expanded aliases contains a leading space. Only normal aliases (not global or suffix aliases) have this behaviour. Note that the command lingers in the internal history until the next command is entered before it vanishes, allowing you to briefly reuse or edit the line. If you want to make it vanish right away without entering another command, type a space and press return.

HIST_LEX_WORDS

By default, shell history that is read in from files is split into words on all white space. This means that arguments with quoted whitespace are not correctly handled, with the consequence that references to words in history lines that have been read from a file may be inaccurate. When this option is set, words read in from a history file are divided up in a similar fashion to normal shell command line handling. Although this produces more accurately delimited words, if the size of the history file is large this can be slow. Trial and error is necessary to decide.

HIST_NO_FUNCTIONS

Remove function definitions from the history list. Note that the function lingers in the internal history until the next command is entered before it vanishes, allowing you to briefly reuse or edit the definition.

HIST_NO_STORE

Remove the history (fc -l) command from the history list when invoked. Note that the command lingers in the internal history until the next command is entered before it vanishes, allowing you to briefly reuse or edit the line.

HIST_REDUCE_BLANKS

Remove superfluous blanks from each command line being added to the history list.

HIST_SAVE_BY_COPY <D>

When the history file is re-written, we normally write out a copy of the file named $HISTFILE.new and then rename it over the old one. However, if this option is unset, we instead truncate the old history file and write out the new version in-place. If one of the history-appending options is enabled, this option only has an effect when the enlarged history file needs to be re-written to trim it down to size. Disable this only if you have special needs, as doing so makes it possible to lose history entries if zsh gets interrupted during the save.

When writing out a copy of the history file, zsh preserves the old file's permissions and group information, but will refuse to write out a new file if it would change the history file's owner.

HIST_SAVE_NO_DUPS

When writing out the history file, older commands that duplicate newer ones are omitted.

HIST_VERIFY

Whenever the user enters a line with history expansion, don't execute the line directly; instead, perform history expansion and reload the line into the editing buffer.

INC_APPEND_HISTORY

This option works like APPEND_HISTORY except that new history lines are added to the $HISTFILE incrementally (as soon as they are entered), rather than waiting until the shell exits. The file will still be periodically re-written to trim it when the number of lines grows 20% beyond the value specified by $SAVE? HIST (see also the HIST_SAVE_BY_COPY option).

INC_APPEND_HISTORY_TIME

This option is a variant of INC_APPEND_HISTORY in which, where possible, the history entry is written out to the file after the command is finished, so that the time taken by the command is recorded correctly in the history file in EXTENDED_HISTORY for? mat. This means that the history entry will not be available immediately from other instances of the shell that are using the same history file.

This option is only useful if INC_APPEND_HISTORY and SHARE_HIS? TORY are turned off. The three options should be considered mu? tually exclusive.

SHARE_HISTORY <K>

This option both imports new commands from the history file, and also causes your typed commands to be appended to the history file (the latter is like specifying INC_APPEND_HISTORY, which

should be turned off if this option is in effect). The history lines are also output with timestamps ala EXTENDED_HISTORY (which makes it easier to find the spot where we left off read? ing the file after it gets re-written).

By default, history movement commands visit the imported lines as well as the local lines, but you can toggle this on and off with the set-local-history zle binding. It is also possible to create a zle widget that will make some commands ignore imported commands, and some include them.

If you find that you want more control over when commands get imported, you may wish to turn SHARE_HISTORY off, INC_AP? PEND_HISTORY or INC_APPEND_HISTORY_TIME (see above) on, and then manually import commands whenever you need them using `fc -RI'.

Initialisation

ALL_EXPORT (-a, ksh: -a)

All parameters subsequently defined are automatically exported.

GLOBAL_EXPORT <Z>

If this option is set, passing the -x flag to the builtins de? clare, float, integer, readonly and typeset (but not local) will also set the -g flag; hence parameters exported to the environ? ment will not be made local to the enclosing function, unless they were already or the flag +g is given explicitly. If the option is unset, exported parameters will be made local in just the same way as any other parameter.

This option is set by default for backward compatibility; it is not recommended that its behaviour be relied upon. Note that the builtin export always sets both the -x and -g flags, and hence its effect extends beyond the scope of the enclosing func? tion; this is the most portable way to achieve this behaviour.

GLOBAL_RCS (-d) <D>

If this option is unset, the startup files /etc/zprofile, /etc/zshrc, /etc/zlogin and /etc/zlogout will not be run. It can be disabled and re-enabled at any time, including inside lo?

cal startup files (.zshrc, etc.).

RCS (+f) <D>

After /etc/zshenv is sourced on startup, source the .zshenv, /etc/zprofile, .zprofile, /etc/zshrc, .zshrc, /etc/zlogin, .zlo? gin, and .zlogout files, as described in the section `Files'. If this option is unset, the /etc/zshenv file is still sourced, but any of the others will not be; it can be set at any time to prevent the remaining startup files after the currently execut? ing one from being sourced.

Input/Output

ALIASES <D>

Expand aliases.

CLOBBER (+C, ksh: +C) <D>

Allows `>' redirection to truncate existing files. Otherwise `>!' or `>|' must be used to truncate a file. If the option is not set, and the option APPEND_CREATE is also not set, `>>!' or `>>|' must be used to create a file. If ei? ther option is set, `>>' may be used.

CORRECT (-0)

Try to correct the spelling of commands. Note that, when the HASH_LIST_ALL option is not set or when some directories in the path are not readable, this may falsely report spelling errors the first time some commands are used. The shell variable CORRECT_IGNORE may be set to a pattern to match words that will never be offered as corrections.

CORRECT_ALL (-O)

Try to correct the spelling of all arguments in a line. The shell variable CORRECT_IGNORE_FILE may be set to a pattern to match file names that will never be offered as corrections.

DVORAK Use the Dvorak keyboard instead of the standard qwerty keyboard as a basis for examining spelling mistakes for the CORRECT and CORRECT_ALL options and the spell-word editor command.

FLOW_CONTROL <D>

If this option is unset, output flow control via start/stop characters (usually assigned to ^S/^Q) is disabled in the shell's editor.

IGNORE_EOF (-7)

Do not exit on end-of-file. Require the use of exit or logout instead. However, ten consecutive EOFs will cause the shell to exit anyway, to avoid the shell hanging if its tty goes away. Also, if this option is set and the Zsh Line Editor is used, widgets implemented by shell functions can be bound to EOF (nor? mally Control-D) without printing the normal warning message. This works only for normal widgets, not for completion widgets.

INTERACTIVE_COMMENTS (-k) <K> <S>

Allow comments even in interactive shells.

HASH_CMDS <D>

Note the location of each command the first time it is executed. Subsequent invocations of the same command will use the saved location, avoiding a path search. If this option is unset, no path hashing is done at all. However, when CORRECT is set, com? mands whose names do not appear in the functions or aliases hash tables are hashed in order to avoid reporting them as spelling errors.

HASH_DIRS <D>

Whenever a command name is hashed, hash the directory containing it, as well as all directories that occur earlier in the path. Has no effect if neither HASH_CMDS nor CORRECT is set.

HASH_EXECUTABLES_ONLY

When hashing commands because of HASH_CMDS, check that the file to be hashed is actually an executable. This option is unset by default as if the path contains a large number of commands, or consists of many remote files, the additional tests can take a long time. Trial and error is needed to show if this option is beneficial.

MAIL_WARNING (-U)

Print a warning message if a mail file has been accessed since the shell last checked.

PATH_DIRS (-Q)

Perform a path search even on command names with slashes in them. Thus if `/usr/local/bin' is in the user's path, and he or she types `X11/xinit', the command `/usr/local/bin/X11/xinit' will be executed (assuming it exists). Commands explicitly be? ginning with `/', `./' or `../' are not subject to the path search. This also applies to the `.' and source builtins.

Note that subdirectories of the current directory are always searched for executables specified in this form. This takes place before any search indicated by this option, and regardless of whether `.' or the current directory appear in the command search path.

PATH_SCRIPT <K> <S>

If this option is not set, a script passed as the first non-op? tion argument to the shell must contain the name of the file to open. If this option is set, and the script does not specify a directory path, the script is looked for first in the current directory, then in the command path. See the section INVOCATION in zsh(1).

PRINT_EIGHT_BIT

Print eight bit characters literally in completion lists, etc.

This option is not necessary if your system correctly returns the printability of eight bit characters (see ctype(3)).

PRINT_EXIT_VALUE (-1)

Print the exit value of programs with non-zero exit status.

This is only available at the command line in interactive shells.

RC_QUOTES

Allow the character sequence `'' to signify a single quote within singly quoted strings. Note this does not apply in quoted strings using the format $'...', where a backslashed sin?

gle quote can be used.

RM_STAR_SILENT (-H) <K> <S>

Do not query the user before executing `rm *' or `rm path/*'.

RM_STAR_WAIT

If querying the user before executing `rm *' or `rm path/*', first wait ten seconds and ignore anything typed in that time. This avoids the problem of reflexively answering `yes' to the query when one didn't really mean it. The wait and query can always be avoided by expanding the `*' in ZLE (with tab).

SHORT_LOOPS <C> <Z>

Allow the short forms of for, repeat, select, if, and function constructs.

SUN_KEYBOARD_HACK (-L)

If a line ends with a backquote, and there are an odd number of backquotes on the line, ignore the trailing backquote. This is useful on some keyboards where the return key is too small, and the backquote key lies annoyingly close to it. As an alterna? tive the variable KEYBOARD_HACK lets you choose the character to be removed.

Job Control

AUTO_CONTINUE

With this option set, stopped jobs that are removed from the job table with the disown builtin command are automatically sent a CONT signal to make them running.

AUTO_RESUME (-W)

Treat single word simple commands without redirection as candi? dates for resumption of an existing job.

BG_NICE (-6) <C> <Z>

Run all background jobs at a lower priority. This option is set by default.

CHECK_JOBS <Z>

Report the status of background and suspended jobs before exit? ing a shell with job control; a second attempt to exit the shell

will succeed. NO_CHECK_JOBS is best used only in combination with NO_HUP, else such jobs will be killed automatically.

The check is omitted if the commands run from the previous com? mand line included a `jobs' command, since it is assumed the user is aware that there are background or suspended jobs. A `jobs' command run from one of the hook functions defined in the section SPECIAL FUNCTIONS in zshmisc(1) is not counted for this purpose.

CHECK_RUNNING_JOBS <Z>

Check for both running and suspended jobs when CHECK_JOBS is en? abled. When this option is disabled, zsh checks only for sus? pended jobs, which matches the default behavior of bash.

This option has no effect unless CHECK_JOBS is set.

HUP <Z>

Send the HUP signal to running jobs when the shell exits.

LONG_LIST_JOBS (-R)

Print job notifications in the long format by default.

MONITOR (-m, ksh: -m)

Allow job control. Set by default in interactive shells.

NOTIFY (-5, ksh: -b) <Z>

Report the status of background jobs immediately, rather than waiting until just before printing a prompt.

POSIX_JOBS <K> <S>

This option makes job control more compliant with the POSIX standard.

When the option is not set, the MONITOR option is unset on entry to subshells, so that job control is no longer active. When the option is set, the MONITOR option and job control remain active in the subshell, but note that the subshell has no access to jobs in the parent shell.

When the option is not set, jobs put in the background or fore? ground with bg or fg are displayed with the same information that would be reported by jobs. When the option is set, only

the text is printed. The output from jobs itself is not af?
fected by the option.
When the option is not set, job information from the parent
shell is saved for output within a subshell (for example, within
a pipeline). When the option is set, the output of jobs is
empty until a job is started within the subshell.
In previous versions of the shell, it was necessary to enable
POSIX_JOBS in order for the builtin command wait to return the
status of background jobs that had already exited. This is no
longer the case.

Prompting

PROMPT_BANG <K>

If set, `!' is treated specially in prompt expansion. See EX?
PANSION OF PROMPT SEQUENCES in zshmisc(1).

PROMPT_CR (+V) <D>

Print a carriage return just before printing a prompt in the
line editor. This is on by default as multi-line editing is
only possible if the editor knows where the start of the line
appears.

PROMPT_SP <D>

Attempt to preserve a partial line (i.e. a line that did not end
with a newline) that would otherwise be covered up by the com?
mand prompt due to the PROMPT_CR option. This works by out?
putting some cursor-control characters, including a series of
spaces, that should make the terminal wrap to the next line when
a partial line is present (note that this is only successful if
your terminal has automatic margins, which is typical).
When a partial line is preserved, by default you will see an in?
verse+bold character at the end of the partial line: a `%' for
a normal user or a `#' for root. If set, the shell parameter
PROMPT_EOL_MARK can be used to customize how the end of partial
lines are shown.
NOTE: if the PROMPT_CR option is not set, enabling this option

will have no effect. This option is on by default.

PROMPT_PERCENT <C> <Z>

If set, `%' is treated specially in prompt expansion. See EX?

PANSION OF PROMPT SEQUENCES in zshmisc(1).

PROMPT_SUBST <K> <S>

If set, parameter expansion, command substitution and arithmetic expansion are performed in prompts. Substitutions within prompts do not affect the command status.

TRANSIENT_RPROMPT

Remove any right prompt from display when accepting a command line. This may be useful with terminals with other cut/paste methods.

Scripts and Functions

ALIAS_FUNC_DEF <S>

By default, zsh does not allow the definition of functions using the `name ()' syntax if name was expanded as an alias: this causes an error. This is usually the desired behaviour, as oth? erwise the combination of an alias and a function based on the same definition can easily cause problems.

When this option is set, aliases can be used for defining func? tions.

For example, consider the following definitions as they might occur in a startup file.

```
alias foo=bar
foo() {
  print This probably does not do what you expect.
}
```

Here, foo is expanded as an alias to bar before the () is en? countered, so the function defined would be named bar. By de? fault this is instead an error in native mode. Note that quot? ing any part of the function name, or using the keyword func? tion, avoids the problem, so is recommended when the function name can also be an alias.

C_BASES

Output hexadecimal numbers in the standard C format, for example `0xFF' instead of the usual `16#FF'. If the option OCTAL_ZEROES is also set (it is not by default), octal numbers will be treated similarly and hence appear as `077' instead of `8#77'. This option has no effect on the choice of the output base, nor on the output of bases other than hexadecimal and octal. Note that these formats will be understood on input irrespective of the setting of C_BASES.

C_PRECEDENCES

This alters the precedence of arithmetic operators to be more like C and other programming languages; the section ARITHMETIC EVALUATION in zshmisc(1) has an explicit list.

DEBUG_BEFORE_CMD <D>

Run the DEBUG trap before each command; otherwise it is run af? ter each command. Setting this option mimics the behaviour of ksh 93; with the option unset the behaviour is that of ksh 88.

ERR_EXIT (-e, ksh: -e)

If a command has a non-zero exit status, execute the ZERR trap, if set, and exit. This is disabled while running initialization scripts.

The behaviour is also disabled inside DEBUG traps. In this case the option is handled specially: it is unset on entry to the trap. If the option DEBUG_BEFORE_CMD is set, as it is by de? fault, and the option ERR_EXIT is found to have been set on exit, then the command for which the DEBUG trap is being exe? cuted is skipped. The option is restored after the trap exits.

Non-zero status in a command list containing && or || is ignored for commands not at the end of the list. Hence

false && true

does not trigger exit.

Exiting due to ERR_EXIT has certain interactions with asynchro? nous jobs noted in the section JOBS in zshmisc(1).

ERR_RETURN

If a command has a non-zero exit status, return immediately from the enclosing function. The logic is similar to that for ERR_EXIT, except that an implicit return statement is executed instead of an exit. This will trigger an exit at the outermost level of a non-interactive script.

Normally this option inherits the behaviour of ERR_EXIT that code followed by `&&' `||' does not trigger a return. Hence in the following:

summit || true

no return is forced as the combined effect always has a zero re? turn status.

Note. however, that if summit in the above example is itself a function, code inside it is considered separately: it may force a return from summit (assuming the option remains set within summit), but not from the enclosing context. This behaviour is different from ERR_EXIT which is unaffected by function scope.

EVAL_LINENO <Z>

If set, line numbers of expressions evaluated using the builtin eval are tracked separately of the enclosing environment. This applies both to the parameter LINENO and the line number output by the prompt escape %i. If the option is set, the prompt es? cape %N will output the string `(eval)' instead of the script or function name as an indication. (The two prompt escapes are typically used in the parameter PS4 to be output when the option XTRACE is set.) If EVAL_LINENO is unset, the line number of the surrounding script or function is retained during the evalua? tion.

EXEC (+n, ksh: +n) <D>

Do execute commands. Without this option, commands are read and checked for syntax errors, but not executed. This option cannot be turned off in an interactive shell, except when `-n' is sup? plied to the shell at startup.

FUNCTION_ARGZERO <C> <Z>

When executing a shell function or sourcing a script, set $0 temporarily to the name of the function/script. Note that tog? gling FUNCTION_ARGZERO from on to off (or off to on) does not change the current value of $0. Only the state upon entry to the function or script has an effect. Compare POSIX_ARGZERO.

LOCAL_LOOPS

When this option is not set, the effect of break and continue commands may propagate outside function scope, affecting loops in calling functions. When the option is set in a calling func? tion, a break or a continue that is not caught within a called function (regardless of the setting of the option within that function) produces a warning and the effect is cancelled.

LOCAL_OPTIONS <K>

If this option is set at the point of return from a shell func? tion, most options (including this one) which were in force upon entry to the function are restored; options that are not re? stored are PRIVILEGED and RESTRICTED. Otherwise, only this op? tion, and the LOCAL_LOOPS, XTRACE and PRINT_EXIT_VALUE options are restored. Hence if this is explicitly unset by a shell function the other options in force at the point of return will remain so. A shell function can also guarantee itself a known shell configuration with a formulation like `emulate -L zsh'; the -L activates LOCAL_OPTIONS.

LOCAL_PATTERNS

If this option is set at the point of return from a shell func? tion, the state of pattern disables, as set with the builtin command `disable -p', is restored to what it was when the func? tion was entered. The behaviour of this option is similar to the effect of LOCAL_OPTIONS on options; hence `emulate -L sh' (or indeed any other emulation with the -L option) activates LO? CAL_PATTERNS.

LOCAL_TRAPS <K>

If this option is set when a signal trap is set inside a func?
tion, then the previous status of the trap for that signal will
be restored when the function exits. Note that this option must
be set prior to altering the trap behaviour in a function; un?
like LOCAL_OPTIONS, the value on exit from the function is ir?
relevant. However, it does not need to be set before any global
trap for that to be correctly restored by a function. For exam?
ple,

```
unsetopt localtraps
trap - INT
fn() { setopt localtraps; trap '' INT; sleep 3; }
```

will restore normal handling of SIGINT after the function exits.

MULTI_FUNC_DEF <Z>

Allow definitions of multiple functions at once in the form `fn1 fn2...()'; if the option is not set, this causes a parse error. Definition of multiple functions with the function keyword is always allowed. Multiple function definitions are not often used and can cause obscure errors.

MULTIOS <Z>

Perform implicit tees or cats when multiple redirections are at? tempted (see the section `Redirection').

OCTAL_ZEROES <S>

Interpret any integer constant beginning with a 0 as octal, per IEEE Std 1003.2-1992 (ISO 9945-2:1993). This is not enabled by default as it causes problems with parsing of, for example, date and time strings with leading zeroes.

Sequences of digits indicating a numeric base such as the `08' component in `08#77' are always interpreted as decimal, regard? less of leading zeroes.

PIPE_FAIL

By default, when a pipeline exits the exit status recorded by the shell and returned by the shell variable $? reflects that of the rightmost element of a pipeline. If this option is set, the

exit status instead reflects the status of the rightmost element of the pipeline that was non-zero, or zero if all elements exited with zero status.

SOURCE_TRACE

If set, zsh will print an informational message announcing the name of each file it loads. The format of the output is similar to that for the XTRACE option, with the message <sourcetrace>. A file may be loaded by the shell itself when it starts up and shuts down (Startup/Shutdown Files) or by the use of the `source' and `dot' builtin commands.

TYPESET_SILENT

If this is unset, executing any of the `typeset' family of commands with no options and a list of parameters that have no values to be assigned but already exist will display the value of the parameter. If the option is set, they will only be shown when parameters are selected with the `-m' option. The option `-p' is available whether or not the option is set.

VERBOSE (-v, ksh: -v)

Print shell input lines as they are read.

XTRACE (-x, ksh: -x)

Print commands and their arguments as they are executed. The output is preceded by the value of $PS4, formatted as described in the section EXPANSION OF PROMPT SEQUENCES in zshmisc(1).

Shell Emulation

APPEND_CREATE <K> <S>

This option only applies when NO_CLOBBER (-C) is in effect. If this option is not set, the shell will report an error when a append redirection (>>) is used on a file that does not already exists (the traditional zsh behaviour of NO_CLOBBER). If the option is set, no error is reported (POSIX behaviour).

BASH_REMATCH

When set, matches performed with the =~ operator will set the BASH_REMATCH array variable, instead of the default MATCH and

match variables. The first element of the BASH_REMATCH array will contain the entire matched text and subsequent elements will contain extracted substrings. This option makes more sense when KSH_ARRAYS is also set, so that the entire matched portion is stored at index 0 and the first substring is at index 1. Without this option, the MATCH variable contains the entire matched text and the match array variable contains substrings.

BSD_ECHO <S>

Make the echo builtin compatible with the BSD echo(1) command. This disables backslashed escape sequences in echo strings un? less the -e option is specified.

CONTINUE_ON_ERROR

If a fatal error is encountered (see the section ERRORS in zsh? misc(1)), and the code is running in a script, the shell will resume execution at the next statement in the script at the top level, in other words outside all functions or shell constructs such as loops and conditions. This mimics the behaviour of in? teractive shells, where the shell returns to the line editor to read a new command; it was the normal behaviour in versions of zsh before 5.0.1.

CSH_JUNKIE_HISTORY <C>

A history reference without an event specifier will always refer to the previous command. Without this option, such a history reference refers to the same event as the previous history ref? erence on the current command line, defaulting to the previous command.

CSH_JUNKIE_LOOPS <C>

Allow loop bodies to take the form `list; end' instead of `do list; done'.

CSH_JUNKIE_QUOTES <C>

Changes the rules for single- and double-quoted text to match that of csh. These require that embedded newlines be preceded by a backslash; unescaped newlines will cause an error message.

In double-quoted strings, it is made impossible to escape `$', ``' or `"' (and `\' itself no longer needs escaping). Command substitutions are only expanded once, and cannot be nested.

CSH_NULLCMD <C>

Do not use the values of NULLCMD and READNULLCMD when running redirections with no command. This make such redirections fail (see the section `Redirection').

KSH_ARRAYS <K> <S>

Emulate ksh array handling as closely as possible. If this option is set, array elements are numbered from zero, an array parameter without subscript refers to the first element instead of the whole array, and braces are required to delimit a subscript (`${path[2]}' rather than just `$path[2]') or to apply modifiers to any parameter (`${PWD:h}' rather than `$PWD:h').

KSH_AUTOLOAD <K> <S>

Emulate ksh function autoloading. This means that when a function is autoloaded, the corresponding file is merely executed, and must define the function itself. (By default, the function is defined to the contents of the file. However, the most common ksh-style case - of the file containing only a simple definition of the function - is always handled in the ksh-compatible manner.)

KSH_OPTION_PRINT <K>

Alters the way options settings are printed: instead of separate lists of set and unset options, all options are shown, marked `on' if they are in the non-default state, `off' otherwise.

KSH_TYPESET

This option is now obsolete: a better appropximation to the behaviour of other shells is obtained with the reserved word interface to declare, export, float, integer, local, readonly and typeset. Note that the option is only applied when the reserved word interface is not in use.

Alters the way arguments to the typeset family of commands, in?

cluding declare, export, float, integer, local and readonly, are processed. Without this option, zsh will perform normal word splitting after command and parameter expansion in arguments of an assignment; with it, word splitting does not take place in those cases.

KSH_ZERO_SUBSCRIPT

Treat use of a subscript of value zero in array or string ex? pressions as a reference to the first element, i.e. the element that usually has the subscript 1. Ignored if KSH_ARRAYS is also set.

If neither this option nor KSH_ARRAYS is set, accesses to an el? ement of an array or string with subscript zero return an empty element or string, while attempts to set element zero of an ar? ray or string are treated as an error. However, attempts to set an otherwise valid subscript range that includes zero will suc? ceed. For example, if KSH_ZERO_SUBSCRIPT is not set,

```
array[0]=(element)
```

is an error, while

```
array[0,1]=(element)
```

is not and will replace the first element of the array.

This option is for compatibility with older versions of the shell and is not recommended in new code.

POSIX_ALIASES <K> <S>

When this option is set, reserved words are not candidates for alias expansion: it is still possible to declare any of them as an alias, but the alias will never be expanded. Reserved words are described in the section RESERVED WORDS in zshmisc(1).

Alias expansion takes place while text is being read; hence when this option is set it does not take effect until the end of any function or other piece of shell code parsed as one unit. Note this may cause differences from other shells even when the op? tion is in effect. For example, when running a command with `zsh -c', or even `zsh -o posixaliases -c', the entire command

argument is parsed as one unit, so aliases defined within the argument are not available even in later lines. If in doubt, avoid use of aliases in non-interactive code.

POSIX_ARGZERO

This option may be used to temporarily disable FUNCTION_ARGZERO and thereby restore the value of $0 to the name used to invoke the shell (or as set by the -c command line option). For com? patibility with previous versions of the shell, emulations use NO_FUNCTION_ARGZERO instead of POSIX_ARGZERO, which may result in unexpected scoping of $0 if the emulation mode is changed in? side a function or script. To avoid this, explicitly enable POSIX_ARGZERO in the emulate command:

emulate sh -o POSIX_ARGZERO

Note that NO_POSIX_ARGZERO has no effect unless FUNCTION_ARGZERO was already enabled upon entry to the function or script.

POSIX_BUILTINS <K> <S>

When this option is set the command builtin can be used to exe? cute shell builtin commands. Parameter assignments specified before shell functions and special builtins are kept after the command completes unless the special builtin is prefixed with the command builtin. Special builtins are ., :, break, con? tinue, declare, eval, exit, export, integer, local, readonly, return, set, shift, source, times, trap and unset.

In addition, various error conditions associated with the above builtins or exec cause a non-interactive shell to exit and an interactive shell to return to its top-level processing.

Furthermore, functions and shell builtins are not executed after an exec prefix; the command to be executed must be an external command found in the path.

Furthermore, the getopts builtin behaves in a POSIX-compatible fashion in that the associated variable OPTIND is not made local to functions.

Moreover, the warning and special exit code from [[ -o non_exis?

tent_option ]] are suppressed.

POSIX_IDENTIFIERS <K> <S>

When this option is set, only the ASCII characters a to z, A to Z, 0 to 9 and _ may be used in identifiers (names of shell parameters and modules).

In addition, setting this option limits the effect of parameter substitution with no braces, so that the expression $# is treated as the parameter $# even if followed by a valid parameter name. When it is unset, zsh allows expressions of the form $#name to refer to the length of $name, even for special variables, for example in expressions such as $#- and $#*.

Another difference is that with the option set assignment to an unset variable in arithmetic context causes the variable to be created as a scalar rather than a numeric type. So after `unset t; (( t = 3 ))'. without POSIX_IDENTIFIERS set t has integer type, while with it set it has scalar type.

When the option is unset and multibyte character support is enabled (i.e. it is compiled in and the option MULTIBYTE is set), then additionally any alphanumeric characters in the local character set may be used in identifiers. Note that scripts and functions written with this feature are not portable, and also that both options must be set before the script or function is parsed; setting them during execution is not sufficient as the syntax variable=value has already been parsed as a command rather than an assignment.

If multibyte character support is not compiled into the shell this option is ignored; all octets with the top bit set may be used in identifiers. This is non-standard but is the traditional zsh behaviour.

POSIX_STRINGS <K> <S>

This option affects processing of quoted strings. Currently it only affects the behaviour of null characters, i.e. character 0 in the portable character set corresponding to US ASCII.

When this option is not set, null characters embedded within strings of the form $'...' are treated as ordinary characters. The entire string is maintained within the shell and output to files where necessary, although owing to restrictions of the li? brary interface the string is truncated at the null character in file names, environment variables, or in arguments to external programs.

When this option is set, the $'...' expression is truncated at the null character. Note that remaining parts of the same string beyond the termination of the quotes are not truncated. For example, the command line argument a$'b\0c'd is treated with the option off as the characters a, b, null, c, d, and with the option on as the characters a, b, d.

POSIX_TRAPS <K> <S>

When this option is set, the usual zsh behaviour of executing traps for EXIT on exit from shell functions is suppressed. In that case, manipulating EXIT traps always alters the global trap for exiting the shell; the LOCAL_TRAPS option is ignored for the EXIT trap. Furthermore, a return statement executed in a trap with no argument passes back from the function the value from the surrounding context, not from code executed within the trap.

SH_FILE_EXPANSION <K> <S>

Perform filename expansion (e.g., ~ expansion) before parameter expansion, command substitution, arithmetic expansion and brace expansion. If this option is unset, it is performed after brace expansion, so things like `~$USERNAME' and `~{pfalstad,rc}' will work.

SH_NULLCMD <K> <S>

Do not use the values of NULLCMD and READNULLCMD when doing redirections, use `:' instead (see the section `Redirection').

SH_OPTION_LETTERS <K> <S>

If this option is set the shell tries to interpret single letter options (which are used with set and setopt) like ksh does.

This also affects the value of the - special parameter.

SH_WORD_SPLIT (-y) <K> <S>

Causes field splitting to be performed on unquoted parameter ex? pansions. Note that this option has nothing to do with word splitting. (See zshexpn(1).)

TRAPS_ASYNC

While waiting for a program to exit, handle signals and run traps immediately. Otherwise the trap is run after a child process has exited. Note this does not affect the point at which traps are run for any case other than when the shell is waiting for a child process.

Shell State

INTERACTIVE (-i, ksh: -i)

This is an interactive shell. This option is set upon initiali? sation if the standard input is a tty and commands are being read from standard input. (See the discussion of SHIN_STDIN.) This heuristic may be overridden by specifying a state for this option on the command line. The value of this option can only be changed via flags supplied at invocation of the shell. It cannot be changed once zsh is running.

LOGIN (-l, ksh: -l)

This is a login shell. If this option is not explicitly set, the shell becomes a login shell if the first character of the argv[0] passed to the shell is a `-'.

PRIVILEGED (-p, ksh: -p)

Turn on privileged mode. Typically this is used when script is to be run with elevated privileges. This should be done as fol? lows directly with the -p option to zsh so that it takes effect during startup.

```
#!/bin/zsh -p
```

The option is enabled automatically on startup if the effective user (group) ID is not equal to the real user (group) ID. In this case, turning the option off causes the effective user and

group IDs to be set to the real user and group IDs. Be aware that if that fails the shell may be running with different IDs than was intended so a script should check for failure and act accordingly, for example:

unsetopt privileged || exit

The PRIVILEGED option disables sourcing user startup files. If zsh is invoked as `sh' or `ksh' with this option set, /etc/suid_profile is sourced (after /etc/profile on interactive shells). Sourcing ~/.profile is disabled and the contents of the ENV variable is ignored. This option cannot be changed using the -m option of setopt and unsetopt, and changing it inside a func? tion always changes it globally regardless of the LOCAL_OPTIONS option.

RESTRICTED (-r)

Enables restricted mode. This option cannot be changed using unsetopt, and setting it inside a function always changes it globally regardless of the LOCAL_OPTIONS option. See the sec? tion `Restricted Shell'.

SHIN_STDIN (-s, ksh: -s)

Commands are being read from the standard input. Commands are read from standard input if no command is specified with -c and no file of commands is specified. If SHIN_STDIN is set explic? itly on the command line, any argument that would otherwise have been taken as a file to run will instead be treated as a normal positional parameter. Note that setting or unsetting this op? tion on the command line does not necessarily affect the state the option will have while the shell is running - that is purely an indicator of whether or not commands are actually being read from standard input. The value of this option can only be changed via flags supplied at invocation of the shell. It can? not be changed once zsh is running.

SINGLE_COMMAND (-t, ksh: -t)

If the shell is reading from standard input, it exits after a

single command has been executed. This also makes the shell non-interactive, unless the INTERACTIVE option is explicitly set on the command line. The value of this option can only be changed via flags supplied at invocation of the shell. It can? not be changed once zsh is running.

Zle

BEEP (+B) <D>

Beep on error in ZLE.

COMBINING_CHARS

Assume that the terminal displays combining characters cor? rectly. Specifically, if a base alphanumeric character is fol? lowed by one or more zero-width punctuation characters, assume that the zero-width characters will be displayed as modifica? tions to the base character within the same width. Not all ter? minals handle this. If this option is not set, zero-width char? acters are displayed separately with special mark-up.

If this option is set, the pattern test [[:WORD:]] matches a zero-width punctuation character on the assumption that it will be used as part of a word in combination with a word character. Otherwise the base shell does not handle combining characters specially.

EMACS If ZLE is loaded, turning on this option has the equivalent ef? fect of `bindkey -e'. In addition, the VI option is unset. Turning it off has no effect. The option setting is not guaran? teed to reflect the current keymap. This option is provided for compatibility; bindkey is the recommended interface.

OVERSTRIKE

Start up the line editor in overstrike mode.

SINGLE_LINE_ZLE (-M) <K>

Use single-line command line editing instead of multi-line. Note that although this is on by default in ksh emulation it only provides superficial compatibility with the ksh line editor and reduces the effectiveness of the zsh line editor. As it has

no effect on shell syntax, many users may wish to disable this option when using ksh emulation interactively.

VI If ZLE is loaded, turning on this option has the equivalent effect of `bindkey -v'. In addition, the EMACS option is unset. Turning it off has no effect. The option setting is not guaranteed to reflect the current keymap. This option is provided for compatibility; bindkey is the recommended interface.

ZLE (-Z)

Use the zsh line editor. Set by default in interactive shells connected to a terminal.

OPTION ALIASES

Some options have alternative names. These aliases are never used for output, but can be used just like normal option names when specifying options to the shell.

BRACE_EXPAND

NO_IGNORE_BRACES (ksh and bash compatibility)

DOT_GLOB

GLOB_DOTS (bash compatibility)

HASH_ALL

HASH_CMDS (bash compatibility)

HIST_APPEND

APPEND_HISTORY (bash compatibility)

HIST_EXPAND

BANG_HIST (bash compatibility)

LOG NO_HIST_NO_FUNCTIONS (ksh compatibility)

MAIL_WARN

MAIL_WARNING (bash compatibility)

ONE_CMD

SINGLE_COMMAND (bash compatibility)

PHYSICAL

CHASE_LINKS (ksh and bash compatibility)

PROMPT_VARS

PROMPT_SUBST (bash compatibility)

STDIN  SHIN_STDIN (ksh compatibility)

TRACK_ALL

HASH_CMDS (ksh compatibility)

SINGLE LETTER OPTIONS

Default set

-0 CORRECT

-1 PRINT_EXIT_VALUE

-2 NO_BAD_PATTERN

-3 NO_NOMATCH

-4 GLOB_DOTS

-5 NOTIFY

-6 BG_NICE

-7 IGNORE_EOF

-8 MARK_DIRS

-9 AUTO_LIST

-B NO_BEEP

-C NO_CLOBBER

-D PUSHD_TO_HOME

-E PUSHD_SILENT

-F NO_GLOB

-G NULL_GLOB

-H RM_STAR_SILENT

-I IGNORE_BRACES

-J AUTO_CD

-K NO_BANG_HIST

-L SUN_KEYBOARD_HACK

-M SINGLE_LINE_ZLE

-N AUTO_PUSHD

-O CORRECT_ALL

-P RC_EXPAND_PARAM

-Q PATH_DIRS

-R LONG_LIST_JOBS

-S REC_EXACT

-T CDABLE_VARS

-U MAIL_WARNING

-V NO_PROMPT_CR

-W AUTO_RESUME

-X LIST_TYPES

-Y MENU_COMPLETE

-Z ZLE

-a ALL_EXPORT

-e ERR_EXIT

-f NO_RCS

-g HIST_IGNORE_SPACE

-h HIST_IGNORE_DUPS

-i INTERACTIVE

-k INTERACTIVE_COMMENTS

-l LOGIN

-m MONITOR

-n NO_EXEC

-p PRIVILEGED

-r RESTRICTED

-s SHIN_STDIN

-t SINGLE_COMMAND

-u NO_UNSET

-v VERBOSE

-w CHASE_LINKS

-x XTRACE

-y SH_WORD_SPLIT

sh/ksh emulation set

-C NO_CLOBBER

-T TRAPS_ASYNC

-X MARK_DIRS

-a ALL_EXPORT

-b NOTIFY

-e ERR_EXIT

-f NO_GLOB

-i INTERACTIVE

-l LOGIN

-m MONITOR

-n NO_EXEC

-p PRIVILEGED

-r RESTRICTED

-s SHIN_STDIN

-t SINGLE_COMMAND

-u NO_UNSET

-v VERBOSE

-x XTRACE

Also note

-A Used by set for setting arrays

-b Used on the command line to specify end of option processing

-c Used on the command line to specify a single command

-m Used by setopt for pattern-matching option setting

-o Used in all places to allow use of long option names

-s Used by set to sort positional parameters