

**Contents**

1	General information, references
2	Grammar (shell syntax)
3	Patterns: globbing and qualifiers
4	Options
5	Options cont.; option aliases, single letter options
6	Expansion: basic forms, history, prompts
7	Expansion: variables: forms and flags
8	Shell variables: set by shell, used by shell
9	Test operators; numeric expressions
10	Completion: contexts, completers, tags
11	Completion cont.: tags cont, styles
12	Completion cont.: styles cont, utility functions
13	Zsh line editor (zle)

**Notes**

The descriptions here are *very* brief. You will not be able to learn shell syntax from them; see the various references below. In particular the completion system is extremely rich and the descriptions of its utility functions are the barest memory joggers.

The start and end of each section is aligned with page boundaries, so you can print out only the parts you want to refer to.

**References**

**Zsh manual:** Supplied with the shell: should be installed in Unix manual page and info formats. Texinfo generates PS or PDF; available as separate doc bundle from same place as the shell.

<http://zsh.sourceforge.net/>: Site with much information about zsh, including HTML manual and a more user-friendly guide to the shell, as well as the FAQ.

**Zsh wiki:** <http://www.zshwiki.org/>: Extensible zsh web pages written by users.

**From Bash to Z Shell: Conquering the Command Line**, by Oliver Kiddle, Jerry Peek and Peter Stephenson, Apress, ISBN 1 59059 376 6. Introduction to interactive use of Unix shells in

general in part 1, concentrating on bash and zsh in parts 2 and 3. The contents of the book are as follows; where noted with page references to this card they expand on the brief hints here.

Part 1 (Introducing the Shell) contains the following chapters:

1	Introduction to Shells
2	Using Shell Features Together
3	More Shell Features (c.f. page 2)

Part 2 (Using bash and zsh) contains the following chapters:

4	Entering and Editing the Command Line (c.f. pages 6 and 13)
5	Starting the Shell (c.f. pages 4 and 5)
6	More About Shell History (c.f. pages 6 and 8)
7	Prompts (c.f. page 6)
8	Files and Directories (c.f. page 9)
9	Pattern Matching (c.f. page 3)
10	Completion (c.f. pages 10 through 12)
11	Jobs and Processes (c.f. page 6)

Part3 (Extending the Shell) contains the following chapters:

12	Variables (c.f. pages 7 and 8)
13	Scripting and Functions (c.f. page 2)
14	Writing Editor Commands (c.f. page 13)
15	Writing Completion Functions (c.f. pages 10 through 12)

The three appendices contain short descriptions of standard Unix

programs, links to external resources, and a glossary.

**Zsh manual pages**

To access documentation from within the shell, use the **man** command with one of the following arguments:

<b>zsh</b>	Introduction, startup and shutdown
<b>zshmisc</b>	Syntax, redirection, functions, jobs, tests
<b>zshexpn</b>	Expansion and substitution
<b>zshparam</b>	Parameters (variables)
<b>zshoptions</b>	Options to the shell
<b>zshbuiltins</b>	Shell builtin commands
<b>zshzle</b>	The line editor, excluding completion
<b>zshcompwid</b>	The low-level completion facilities
<b>zshcompsys</b>	The new completion system (more readable)
<b>zshcompctl</b>	The old completion system (deprecated)
<b>zshmodules</b>	Modules loadable with <b>zmodload</b>
<b>zshtcpsys</b>	Functions for using raw TCP via builtins
<b>zshzftpsys</b>	Functions for using FTP via builtins
<b>zshcontrib</b>	Contributed functions for zle etc.
<b>zshall</b>	Everything in one large manual page

**Mailing lists**

[zsh-users@zsh.org](mailto:zsh-users@zsh.org): users' mailing list for general questions and tips; to join, mail [zsh-users-subscribe@zsh.org](mailto:zsh-users-subscribe@zsh.org).

[zsh-workers@zsh.org](mailto:zsh-workers@zsh.org): mailing list for bug reports, patches and developers' discussions; to join, mail [zsh-workers-subscribe@zsh.org](mailto:zsh-workers-subscribe@zsh.org). New developers with some Unix/Linux experience are welcome.

**Grammar**

*List* is any sequence of *sublists* (including just one) separated by **;** or **newline**. **;** and **newline** are always interchangeable except in **;;**.

*Sublist* is any sequence of *pipelines* (including just one) connected by **&&** or **|**.

*Pipeline* is any sequence of simple commands connected by **|**.

*Command* is either a simple command (a command *word*) followed optionally by *word* ... or one of the special commands below.

*Word* is any text that produces a single word when expanded; *word* ... is any number of these separated by whitespace.

*Name* is a shell identifier: an alphabetic character or **\_** followed by any sequence of alphanumeric characters or **\_**.

[ ... ] indicates optional; dots on their own line mean any number of repetitions of the line just above.

Bold text is to be typed literally.

Status “true” or “false” is determined by: for commands, the return status; for pipelines the last command; for sublists the last pipeline; for lists the last sublist that was executed.

**sublist1 && sublist2 [ && sublist3 ... ]**

Execute *sublists* until one is false.

**sublist1 || sublist2 [ || sublist2 ... ]**

Execute *sublists* until one is true. Note strings of **&&** sublists can contain **||** sublists and vice versa; they are parsed left to right.

**command1 | command2 [ | command3 ... ]**

Execute *command1*, sending its output to the input of *command2*, and so on (a *pipeline*).

```
if listi1; then[;] listt1;
[ elif listi2; then listt2; ]
```

...

```
[ else listt3; ]
```

```
fi
```

If *listi1* is true, execute *listt1*; else if *listi2* is true execute *listt2*; else execute *listt3*.

```
for name [ in word ... ]
```

```
do list;
```

```
done
```

Execute *list* with variable *name* set to each of *word* ... in turn

If *in* ... is omitted the positional parameters are used.

```
for name in word ...; { list }
```

```
foreach name ( word ... ) [;]
```

```
list;
```

```
end
```

Non-portable alternative forms.

```
while listw; do listd; done
```

While *listw* is true execute *listd*.

```
until listu; do listd; done
```

Non-portable: while *listu* is not true execute *listd*.

```
repeat numexp; do list; done
```

```
repeat numexp sublist
```

Non-portable: repeat *list* or *sublist* *numexp* times.

```
case word in
```

```
[(| pattern1[|pattern2...]) [|;] list ;;
```

...

```
esac
```

Try matching word against every pattern in turn until success.

Execute the corresponding list. **;&** instead of **&&** means fall through to next *list*.

```
case word {
```

```
[(| pattern1[|pattern2...]) [|;] list ;;
```

...

```
}
```

Non-portable alternative.

```
select name [ in word ...];
```

```
do list;
```

```
done
```

Print menu of *words*, read a number, set *name* to selected word, execute *list* until end of input. Portable but rare.

```
(list[;])
```

Execute *list* in a subshell (a new process where nothing that happens affects the current shell).

```
{list[;]}
```

Execute *list* (no new process: simply separates list from what's around and can take redirections).

```
function nameword {[;] list[;] }
```

```
nameword () {[;] list[;] }
```

Define function named *nameword*; executes list when run; running *nameword word1* ... makes *word1* ... available as **\$1** etc. in function body. *list* must end with **;** or newline for portability. *nameword* can be repeated to define multiple functions (rare, non-portable).

```
time [ pipeline ]
```

Report time for *pipeline* if given else totals for current shell.

```
[[ condition ]]
```

Evaluate *condition* (see below), gives status true or false.

**Pattern matching (globbing)**

Basic patterns:

<b>*</b>	Any string
<b>?</b>	Any character
<b>[class]</b>	Any single character from <i>class</i>
<b>[^class]</b>	Any single character not from <i>class</i>
<b>&lt;num1-num2&gt;</b>	Any number between <i>num1</i> and <i>num2</i>
	<b>&lt;-num2&gt;</b> from 0; <b>&lt;num1-&gt;</b> to infinity.
<b>**/</b>	Directories to any level
<b>(pat1)</b>	Group patterns
<b>(pat1 pat2)</b>	<i>pat1</i> or <i>pat2</i> (any number of  's)

Character classes may contain any character or the following special patterns in any mix; literal `-` must be first; literal `^` must not be first:

<b>a-b</b>	A character in the range <i>a</i> to <i>b</i>
<b>[:alnum:]</b>	An alphanumeric character
<b>[:alpha:]</b>	An alphabetic character
<b>[:ascii:]</b>	A character in the ASCII character set
<b>[:blank:]</b>	A space or tab
<b>[:cntrl:]</b>	A control character
<b>[:digit:]</b>	A decimal digit
<b>[:graph:]</b>	A printable character other than whitespace
<b>[:lower:]</b>	A lower case letter
<b>[:print:]</b>	A printable character
<b>[:punct:]</b>	A punctuation character
<b>[:space:]</b>	Any whitespace character
<b>[:upper:]</b>	An upper case letter
<b>[:xdigit:]</b>	A hexadecimal digit

Extended patterns (option **EXTENDED\_GLOB** must be set):

<b>^pat</b>	Anything that doesn't match <i>pat</i>
<b>pat1^pat2</b>	Match <i>pat1</i> then anything other than <i>pat2</i>
<b>pat1~pat2</b>	Anything matching <i>pat1</i> but not <i>pat2</i>
<b>X#</b>	Zero or more occurrences of element <i>X</i>
<b>X##</b>	One or more occurrences of element <i>X</i>

**KSH\_GLOB** operators (patterns may contain | for alternatives):

<b>@(pat)</b>	Group patterns
<b>*(pat)</b>	Zero or more occurrences of <i>pat</i>
<b>+(pat)</b>	One or more occurrences of <i>pat</i>
<b>?(pat)</b>	Zero or one occurrences of <i>pat</i>
<b>!(pat)</b>	Anything but the pattern <i>pat</i>

Globbing flags with **EXTENDED\_GLOB**:

<b>(#i)</b>	Match case insensitively
<b>(#l)</b>	Lower case matches upper case
<b>(#I)</b>	Match case sensitively
<b>(#b)</b>	Parentheses set <b>match</b> , <b>mbegin</b> , <b>mend</b>
<b>(#B)</b>	Parentheses no longer set arrays
<b>(#m)</b>	Match in <b>MATCH</b> , <b>MBEGIN</b> , <b>MEND</b>
<b>(#M)</b>	Don't use <b>MATCH</b> etc.
<b>(#anum)</b>	Match with <i>num</i> approximations
<b>(#s)</b>	Match only at start of test string
<b>(#e)</b>	Match only at end of test string
<b>(#qexpr)</b>	<i>expr</i> is a set of glob qualifiers (below)

Glob qualifiers (in parentheses after file name pattern):

<b>/</b>	Directory
<b>F</b>	Non-empty directory; for empty use <b>(/^F)</b>
<b>.</b>	Plain file
<b>@</b>	Symbolic link
<b>=</b>	Socket
<b>p</b>	Name pipe (FIFO)
<b>*</b>	Executable plain file
<b>%</b>	Special file
<b>%b</b>	Block special file
<b>%c</b>	Character special file
<b>r</b>	Readable by owner (N.B. not current user)
<b>w</b>	Writable by owner
<b>x</b>	Executable by owner
<b>A</b>	Readable by members of file's group
<b>I</b>	Writable by members of file's group
<b>E</b>	Executable by members of file's group
<b>R</b>	World readable
<b>W</b>	World writable
<b>X</b>	World executable
<b>s</b>	Setuid

<b>S</b>	Setgid
<b>t</b>	Sticky bit
<b>fspec</b>	Has <b>chmod</b> -style permissions <i>spec</i>
<b>estring</b>	Evaluation <i>string</i> returns true status
<b>+cmd</b>	Same but <i>cmd</i> must be alphanumeric or <code>_</code>
<b>ddev</b>	Device number <i>dev</i> (major*256 + minor)
<b>l[-+]num</b>	Link count is (less than, greater than) <i>num</i>
<b>U</b>	Owned by current effective UID
<b>G</b>	Owned by current effective GID
<b>uuid</b>	Owned by given <i>uid</i> (may be <i>&lt;name&gt;</i> )
<b>ggid</b>	Owned by given <i>gid</i> (may be <i>&lt;name&gt;</i> )
<b>a[Mwhms][-+]n</b>	Access time in given units (see below)
<b>m[Mwhms][-+]n</b>	Modification time in given units
<b>c[Mwhms][-+]n</b>	Inode change time in given units
<b>L[kmp][-+]n</b>	Size in given units (see below)
<b>^</b>	Negate following qualifiers
<b>-</b>	Toggle following links (first one turns on)
<b>M</b>	Mark directories
<b>T</b>	Mark directories, links, special files
<b>N</b>	Whole pattern expands to empty if no match
<b>D</b>	Leading dots may be matched
<b>n</b>	Sort numbers numerically
<b>o[nLlamcd]</b>	Order by given code (as below; may repeat)
<b>O[nLlamcd]</b>	Order by reverse of given code
<b>[num]</b>	Select <i>num</i> th file in current order
<b>[num1, num2]</b>	Select <i>num1</i> th to <i>num2</i> th file (as arrays)
<b>:X</b>	History modifier <i>X</i> ; may have more

Time units are Month, week, hour, minute, second; default is day.  
Size units are kilobytes, megabytes or 512-byte blocks (**p**); default is bytes; upper case means the same as lower case.  
Order codes are name (default), size, link count, access time, modification time, inode change time, directory depth.

Options			
Set options with <b>setopt</b> , unset with <b>unsetopt</b> . Asterisk indicates on by default for native zsh.			
* <b>ALIASES</b>	Expand aliases	<b>CSH_NULL_GLOB</b>	One glob must succeed, failures go
<b>ALL_EXPORT</b>	Export all variables to environment	<b>DVORAK</b>	Dvorak keyboard for correction
* <b>ALWAYS_LAST_PROMPT</b>	Completion lists after prompt	<b>EMACS</b>	Same as <b>bindkey -e</b>
<b>ALWAYS_TO_END</b>	On completion go to end of word	* <b>EQUALS</b>	Expand <b>=cmd</b> to <b>/path/to/cmd</b>
* <b>APPEND_HISTORY</b>	History appends to existing file	<b>ERR_EXIT</b>	Exit shell on non-zero status
<b>AUTO_CD</b>	Directory as command does <b>cd</b>	<b>ERR_RETURN</b>	Return from function instead
<b>AUTO_CONTINUE</b>	Jobs are continued when <b>disowned</b>	* <b>EVAL_LINE_NO</b>	<b>\$LINENO</b> counts inside <b>eval</b> code
* <b>AUTO_LIST</b>	List ambiguous completions	* <b>EXEC</b>	Execute commands
* <b>AUTO_MENU</b>	Menu complete after two tabs	<b>EXTENDED_GLOB</b>	See globbing section above
<b>AUTO_NAME_DIRS</b>	Variables always can be %~ abbrevs	<b>EXTENDED_HISTORY</b>	Timestamps saved to history file
* <b>AUTO_PARAM_KEYS</b>	Magic completion for parameters	* <b>FLOW_CONTROL</b>	Use ^S/^Q style flow control
* <b>AUTO_PARAM_SLASH</b>	<b>\$dirname</b> completes with /	* <b>FUNCTION_ARGZERO</b>	<b>\$0</b> in function is its name
<b>AUTO_PUSHD</b>	<b>cd</b> uses directory stack too	* <b>GLOB</b>	Use globbing as described above
* <b>AUTO_REMOVE_SLASH</b>	Trailing / in completion removed	* <b>GLOBAL_EXPORT</b>	Exported variables not made local
<b>AUTO_RESUME</b>	<b>cmd</b> can resume job <b>%cmd</b>	* <b>GLOBAL_RCS</b>	Execute <b>/etc/z*</b> files
* <b>BAD_PATTERN</b>	Errors on pattern syntax; else literal	<b>GLOB_ASSIGN</b>	<b>var=*</b> expands, assigns array
* <b>BANG_HIST</b>	! style history allowed	<b>GLOB_COMPLETE</b>	Patterns are active in completion
* <b>BARE_GLOB_QUAL</b>	Glob qualifiers with bare parens	<b>GLOB_DOTS</b>	Patterns may match leading dots
<b>BASH_AUTO_LIST</b>	List completions on second tab	<b>GLOB_SUBST</b>	Substituted characters may glob
* <b>BEEP</b>	Beep on all errors	* <b>HASH_CMDS</b>	Store command location for speed
* <b>BG_NICE</b>	Background jobs at lower priority	* <b>HASH_DIRS</b>	Store for all commands in dir
<b>BRACE_CCL</b>	<b>X{ab}</b> expands to <b>Xa Xb</b>	* <b>HASH_LIST_ALL</b>	Store all on first completion
<b>BSD_ECHO</b>	No echo escapes unless <b>-e</b> given	<b>HIST_ALLOW_CLOBBER</b>	On clobber error, up arrow to retry
* <b>CASE_GLOB</b>	Glob case sensitively	* <b>HIST_BEEP</b>	Beep when going beyond history
<b>C_BASES</b>	Output hexadecimal with <b>0x</b>	<b>HIST_EXPIRE_DUPS_FIRST</b>	Duplicate history entries lost first
<b>CDABLE_VARS</b>	<b>cd var</b> works if <b>\$var</b> is directory	<b>HIST_FIND_NO_DUPS</b>	History search finds once only
<b>CHASE_DOTS</b>	Resolve <b>..</b> in <b>cd</b>	<b>HIST_IGNORE_ALL_DUPS</b>	Remove all earlier duplicate lines
<b>CHASE_LINKS</b>	Resolve symbolic links in <b>cd</b>	<b>HIST_IGNORE_DUPS</b>	Remove duplicate of previous line
* <b>CHECK_JOBS</b>	Check jobs before exiting shell	<b>HIST_IGNORE_SPACE</b>	Don't store lines starting with space
* <b>CLOBBER</b>	Allow redirections to overwrite	<b>HIST_NO_FUNCTIONS</b>	Don't store shell functions
<b>COMPLETE_ALIASES</b>	Completion uses unexpanded aliases	<b>HIST_NO_STORE</b>	Don't store <b>history</b> and <b>fc</b>
<b>COMPLETE_IN_WORD</b>	Completion works inside words	<b>HIST_REDUCE_BLANKS</b>	Trim multiple insignificant blanks
<b>CORRECT</b>	Correct spelling of commands	<b>HIST_SAVE_NO_DUPS</b>	Remove duplicates when saving
<b>CORRECT_ALL</b>	Correct spelling of all arguments	<b>HIST_VERIFY</b>	Show ! history line for editing
<b>CSH_JUNKIE_HISTORY</b>	Single ! for previous command	* <b>HUP</b>	Send <b>SIGHUP</b> to processes on exit
<b>CSH_JUNKIE_LOOPS</b>	<b>list; end</b> for <b>do...done</b>	<b>IGNORE_BRACES</b>	Don't use <b>{a,b}</b> expansions
<b>CSH_JUNKIE_QUOTES</b>	No newlines in quotes	<b>IGNORE_EOF</b>	Ignore ^D ( <b>stty eof</b> char)
<b>CSH_NULLCMD</b>	Redirections with no commands fail	<b>INC_APPEND_HISTORY</b>	Save history line by line
		<b>INTERACTIVE</b>	Shell is interactive
		<b>INTERACTIVE_COMMENTS</b>	# on interactive line for comment
		<b>KSH_ARRAYS</b>	Indexing etc. for arrays like ksh
		<b>KSH_AUTOLOAD</b>	Function file includes function name
		<b>KSH_GLOB</b>	See globbing above
		<b>KSH_OPTION_PRINT</b>	Show all options plus on or off
		<b>KSH_TYPESET</b>	No word splitting in <b>typeset</b> etc.
		* <b>LIST_AMBIGUOUS</b>	List completions when ambiguous
		* <b>LIST_BEEP</b>	Beep on ambiguous completion
		<b>LIST_PACKED</b>	More compact completion lists
		<b>LIST_ROWS_FIRST</b>	List completions across
		* <b>LIST_TYPES</b>	File types listed in completion
		<b>LOCAL_OPTIONS</b>	Options reset on function return
		<b>LOCAL_TRAPS</b>	Traps reset on function return
		<b>LOGIN</b>	Shell is login shell
		<b>LONG_LIST_JOBS</b>	More verbose listing of jobs
		<b>MAGIC_EQUAL_SUBST</b>	Special expansion after all =
		<b>MAIL_WARNING</b>	Warn if mail file timestamp changed
		<b>MARK_DIRS</b>	Append / to globbed directories
		<b>MENU_COMPLETE</b>	Cycle through ambiguous matches
		<b>MONITOR</b>	Shell has job control enabled
		* <b>MULTIOS</b>	Multiple redirections are special
		* <b>NOMATCH</b>	Error if glob fails to match
		* <b>NOTIFY</b>	Asynchronous job control messages
		<b>NULL_GLOB</b>	Failed globs are removed from line
		<b>NUMERIC_GLOB_SORT</b>	Numbers in globs sorted numerically
		<b>OCTAL_ZEROES</b>	Leading zeros in integers force octal
		<b>OVERSTRIKE</b>	Start line editor in overstrike mode
		<b>PATH_DIRS</b>	<b>dir/cmd</b> can be found in <b>\$PATH</b>
		<b>POSIX_BUILTINS</b>	Illogical command behaviour
		<b>PRINT_EIGHT_BIT</b>	Print all 8-bit characters directly
		<b>PRINT_EXIT_VALUE</b>	Return status printed unless zero
		<b>PRIVILEGED</b>	Special behaviour on <b>setuid/setgid</b>
		<b>PROMPT_BANG</b>	Special treatment of ! in prompt
		* <b>PROMPT_CR</b>	Prompt always at start of line
		* <b>PROMPT_PERCENT</b>	% escapes expanded in prompts
		<b>PROMPT_SUBST</b>	\$ expansion etc. in prompts
		<b>PUSHD_IGNORE_DUPS</b>	Don't push dir multiply on stack
		<b>PUSHD_MINUS</b>	Reverse sense of - and + in <b>pushd</b>
		<b>PUSHD_SILENT</b>	No non-err messages from <b>pushd</b>
		<b>PUSHD_TO_HOME</b>	<b>pushd</b> with no argument goes to ~

<b>RC_EXPAND_PARAM</b>	<b>X\$array</b> gives <b>Xelt1 Xelt2</b> etc.
<b>RC_QUOTES</b>	' ' inside single quotes gives ' '
<b>*RCS</b>	Run startup files
<b>REC_EXACT</b>	Exact completion matches are good
<b>RESTRICTED</b>	Shell has restricted capabilities
<b>RM_STAR_SILENT</b>	Don't warn on <b>rm *</b>
<b>RM_STAR_WAIT</b>	Wait before asking if <b>rm *</b> is OK
<b>SHARE_HISTORY</b>	Save and restore history per line
<b>SH_FILE_EXPANSION</b>	~ etc. expansion done early
<b>SH_GLOB</b>	Disables non-extended zsh globs
<b>SHIN_STDIN</b>	Shell input comes from stdin
<b>SH_NULL_CMD</b>	Commandless redirections like <b>sh</b>
<b>SH_OPTION_LETTERS</b>	Single letter options are like <b>sh</b>
<b>*SHORT_LOOPS</b>	<b>for words; list</b> works
<b>SH_WORD_SPLIT</b>	Split non-array variables yuckily
<b>SINGLE_COMMAND</b>	Execute one command then exit
<b>SINGLE_LINE_ZLE</b>	Line editing on single line (bad tty)
<b>SUN_KEYBOARD_HACK</b>	Unmatched ` at end of line ignored
<b>TRANSIENT_RPROMPT</b>	Right prompt goes away after edit
<b>TRAPS_ASYNC</b>	Traps may run when <b>waiting</b>
<b>TYPESET_SILENT</b>	Silent on <b>typeset foo</b>
<b>*UNSET</b>	Unset variables OK, treat as empty
<b>VERBOSE</b>	Output commands to be executed
<b>VI</b>	Same as <b>bindkey -v</b>
<b>XTRACE</b>	Show trace of execution with <b>\$PS4</b>
<b>ZLE</b>	Line editor used to input lines

Option aliases (native zsh on right):

<b>BRACE_EXPAND</b>	<b>NO_IGNORE_BRACES</b>
<b>DOT_GLOB</b>	<b>GLOB_DOTS</b>
<b>HASH_ALL</b>	<b>HASH_CMDS</b>
<b>HIST_APPEND</b>	<b>APPEND_HISTORY</b>
<b>HIST_EXPAND</b>	<b>BANG_HIST</b>
<b>LOG</b>	<b>NO_HIST_NO_FUNCTIONS</b>
<b>MAIL_WARN</b>	<b>MAIL_WARNING</b>
<b>ONE_CMD</b>	<b>SINGLE_COMMAND</b>
<b>PHYSICAL</b>	<b>CHASE_LINKS</b>
<b>PROMPT_VARS</b>	<b>PROMPT_SUBST</b>
<b>STDIN</b>	<b>SHIN_STDIN</b>
<b>TRACK_ALL</b>	<b>HASH_CMDS</b>

Single letter options (used with **set** as well as **setopt**):

<b>-0</b>	<b>CORRECT</b>
<b>-1</b>	<b>PRINT_EXIT_VALUE</b>
<b>-2</b>	<b>NO_BAD_PATTERN</b>
<b>-3</b>	<b>NO_NO_MATCH</b>
<b>-4</b>	<b>GLOB_DOTS</b>
<b>-5</b>	<b>NOTIFY</b>
<b>-6</b>	<b>BG_NICE</b>
<b>-7</b>	<b>IGNORE_EOF</b>
<b>-8</b>	<b>MARK_DIRS</b>
<b>-9</b>	<b>AUTO_LIST</b>
<b>-B</b>	<b>NO_BEEP</b>
<b>-C</b>	<b>NO_CLOBBER</b>
<b>-D</b>	<b>PUSHD_TO_HOME</b>
<b>-E</b>	<b>PUSHD_SILENT</b>
<b>-F</b>	<b>NO_GLOB</b>
<b>-G</b>	<b>NULL_GLOB</b>
<b>-H</b>	<b>RM_STAR_SILENT</b>
<b>-I</b>	<b>IGNORE_BRACES</b>
<b>-J</b>	<b>AUTO_CD</b>
<b>-K</b>	<b>NO_BANG_HIST</b>
<b>-L</b>	<b>SUN_KEYBOARD_HACK</b>
<b>-M</b>	<b>SINGLE_LINE_ZLE</b>
<b>-N</b>	<b>AUTO_PUSHD</b>
<b>-O</b>	<b>CORRECT_ALL</b>
<b>-P</b>	<b>RC_EXPAND_PARAM</b>
<b>-Q</b>	<b>PATH_DIRS</b>
<b>-R</b>	<b>LONG_LIST_JOBS</b>
<b>-S</b>	<b>REC_EXACT</b>
<b>-T</b>	<b>CDABLE_VARS</b>
<b>-U</b>	<b>MAIL_WARNING</b>
<b>-V</b>	<b>NO_PROMPT_CR</b>
<b>-W</b>	<b>AUTO_RESUME</b>
<b>-X</b>	<b>LIST_TYPES</b>
<b>-Y</b>	<b>MENU_COMPLETE</b>
<b>-Z</b>	<b>ZLE</b>
<b>-a</b>	<b>ALL_EXPORT</b>
<b>-e</b>	<b>ERR_EXIT</b>
<b>-f</b>	<b>NO_RCS</b>
<b>-g</b>	<b>HIST_IGNORE_SPACE</b>
<b>-h</b>	<b>HIST_IGNORE_DUPS</b>
<b>-i</b>	<b>INTERACTIVE</b>
<b>-k</b>	<b>INTERACTIVE_COMMENTS</b>

<b>-l</b>	<b>LOGIN</b>
<b>-m</b>	<b>MONITOR</b>
<b>-n</b>	<b>NO_EXEC</b>
<b>-p</b>	<b>PRIVILEGED</b>
<b>-r</b>	<b>RESTRICTED</b>
<b>-s</b>	<b>SHIN_STDIN</b>
<b>-t</b>	<b>SINGLE_COMMAND</b>
<b>-u</b>	<b>NO_UNSET</b>
<b>-v</b>	<b>VERBOSE</b>
<b>-w</b>	<b>CHASE_LINKS</b>
<b>-x</b>	<b>XTRACE</b>
<b>-y</b>	<b>SH_WORD_SPLIT</b>

Note also **-A** to set arrays, **-b** to end option processing, **-c** to pass a single command, **-m** to set pattern argument, **-o** to specify long name (may repeat), **-s** to sort positional parameters.

**Expansion**

Basic forms of expansion in the order they order:

<b>!expr</b>	History expansion
<b>alias</b>	Alias expansion
<b>&lt;(cmds)</b>	Replaced by file with output from <i>cmds</i>
<b>=(cmds)</b>	Same but can be reread (use for <b>diff</b> )
<b>&gt;(cmds)</b>	Replaced by file with input to <i>cmds</i>
<b>\$var</b>	Variable substitution
<b>\${var}</b>	Same but protected, allows more options
<b>\$(cmds)</b>	Replaced by output of <i>cmds</i>
<b>`cmds`</b>	Older form of same, harder to nest
<b>\$( (expr) )</b>	Arithmetic result of evaluating <i>expr</i>
<b>X{a,b}Y</b>	<i>XaY XbY</i> (N.B. does no pattern matching)
<b>X{1..3}Y</b>	<i>X1Y X2Y X3Y</i>
<b>X{08..10}Y</b>	<i>X08Y X09Y X10Y</i>
<b>~user, ~dir</b>	User home, named dir ( <i>dir</i> is var name)
<b>=cmd</b>	/full/path/to/cmd
<b>pattern</b>	Glob file names, as above

History expansion:

<b>!!</b>	Immediately preceding line (all of it)
<b>!{!}</b>	Same but protected, may have args in { }
<b>!</b>	Line just referred to, default <b>!!</b>
<b>!13</b>	Line numbered 13 ( <b>history</b> shows nos.)
<b>!-2</b>	Command two before current
<b>!cmd</b>	Last command beginning <i>cmd</i>
<b>!?str</b>	Last command containing <i>str</i>
<b>!#</b>	Current command line so far

Word selectors:

<b>!!:0</b>	Extract argument 0 (command word)
<b>!!:1</b>	Argument numbered 1 (first cmd arg)
<b>!!:^</b>	Also argument 1
<b>!:\$</b>	Last command argument
<b>!:%</b>	Word found by <b>!?str</b> (needs correct line)
<b>!!:2-4</b>	Word 2 to 4 inclusive
<b>!!: -4</b>	Words 0 to 4 inclusive
<b>!!:*</b>	Words 1 to \$ inclusive
<b>!!:2*</b>	Words 2 to \$ inclusive
<b>!!:2-</b>	Words 2 to \$-1 inclusive

Modifiers on arguments (can omit word selector):

<b>!!:1:h</b>	Trailing path component removed
<b>!!:1:t</b>	Only trailing path component left
<b>!!:1:r</b>	File extension .ext removed
<b>!!:1:e</b>	Only extension ext left
<b>!!:1:p</b>	Print result but don't execute
<b>!!:1:q</b>	Quote from further substitution
<b>!!:1:Q</b>	Strip one level of quotes
<b>!!:1:x</b>	Quote and also break at whitespace
<b>!!:1:l</b>	Convert to all lower case
<b>!!:1:u</b>	Convert to all upper case
<b>!!:1:s/s1/s2/</b>	Replace string <i>s1</i> by <i>s2</i>
<b>!!:1:gs/s2/s2/</b>	Same but global
<b>!!:1:&amp;</b>	Use same <i>s1</i> and <i>s2</i> on new target

Most modifiers work on variables (e.g. **\${var:h}**) or in glob qualifiers (e.g. **\*( :h)**), the following only work there:

<b>\${var:fm}</b>	Repeat modifier <i>m</i> till stops changing
<b>\${var:F:N:m}</b>	Same but no more than <i>N</i> times
<b>\${var:wm}</b>	Apply modifier <i>m</i> to words of string
<b>\${var:W:sep:m}</b>	Same but words are separated by <i>sep</i>

Prompt expansion (with **PROMPT\_PERCENT**, on by default); may take a decimal number *n* (default 0) immediately after the %:

<b>%! %h</b>	Current history event number
<b>%#</b>	# if superuser, else %
<b>%%</b>	A single %
<b>%)</b>	A ) (use with <b>%X(.tstr.fstr)</b> )
<b>%*</b>	Time in 24-hour format with seconds
<b>%/ %d</b>	<b>\$PWD</b> ; <i>n</i> gives trailing parts, <b>-n</b> leading
<b>%c %. %C</b>	Deprecated alternatives, differ by default <i>n</i>
<b>%?</b>	Return status of last command
<b>%@ %t</b>	Time of day in am/pm format
<b>%B (%b)</b>	Start (stop) bold face mode
<b>%D %D{str}</b>	Date as <b>YY-MM-DD</b> , optional strftime spec
<b>%E</b>	Clear to end of line
<b>%i</b>	Script/function line number ( <b>\$LINENO</b> )
<b>%j</b>	Number of jobs as listed by jobs
<b>%L</b>	Shell depth ( <b>\$SHLVL</b> )
<b>%l</b>	Login terminal without <b>/dev</b> or <b>/dev/tty</b>

<b>%M</b>	Full host name
<b>%m</b>	Host name to first dot or <i>n</i> dots
<b>%N</b>	Name of script, function, sourced file
<b>%n</b>	Name of user (same as <b>\$USERNAME</b> )
<b>%S %s</b>	Start (stop) standout mode
<b>%T</b>	Time of day, 24-hour format
<b>%U %u</b>	Start (stop) underline mode (patchy support)
<b>%v</b>	<i>n</i> th component of <b>\$psvar</b> array
<b>%W</b>	Date as middle-endian <b>MM/DD/YY</b>
<b>%w</b>	Date as <b>DAY DD</b>
<b>%y</b>	Login terminal without <b>/dev</b>
<b>%_</b>	Parser state (continuation lines, debug)
<b>%~</b>	Like <b>%/</b> , <b>%d</b> but with tilde substitution
<b>%{esc%}</b>	Escape sequence <b>esc</b> doesn't move cursor
<b>%X(.tstr.fstr)</b>	<i>tstr</i> if test <i>X</i> gives <i>n</i> , else <i>fstr</i>
<b>%&lt;str&lt;</b>	Truncate to <i>n</i> on left, <i>str</i> on left if so
<b>%&gt;str&gt;</b>	Truncate to <i>n</i> on right, <i>str</i> on right if so

Test characters in **%X(.tstr.fstr)**: **!** Privileged; **#** uid *n*; **?** last status *n*; **\_** at least *n* nested constructs; **/** at least *n* **\$PWD** elements; **~** same with **~** subst; **D** month is *n*; **d** day of month is *n*; **g** effective gid is *n*; **j** at least *n* jobs; **L** **\$SHLVL** at least *n*; **l** at least *n* chars on line so far; **S** **\$SECONDS** at least *n*; **T** hours is *n*; **t** minutes is *n*; **v** at least *n* components in **\$psvar**; **w** day of week is *n* (Sunday = 0).

**Parameter (Variable) Expansion**

Basic forms: *str* will also be expanded; most forms work on words of array separately:

<b>\${var}</b>	Substitute contents of <i>var</i> , no splitting
<b>\${+var}</b>	1 if <i>var</i> is set, else 0
<b>\${var:-str}</b>	<i>var</i> if non-null, else <i>str</i>
<b>\${var-str}</b>	<i>var</i> if set (even if null) else <i>str</i>
<b>\${var:=str}</b>	<i>var</i> if non-null, else <i>str</i> and set <i>var</i> to it
<b>\${var::=str}</b>	Same but always use <i>str</i>
<b>\${var:?str}</b>	<i>var</i> if non-null else error, abort
<b>\${var:+str}</b>	<i>str</i> if <i>var</i> is non-null
<b>\${var#pat}</b>	min match of <i>pat</i> removed from head
<b>\${var##pat}</b>	max match of <i>pat</i> removed from head
<b>\${var%pat}</b>	min match of <i>pat</i> removed from tail
<b>\${var%%pat}</b>	max match of <i>pat</i> removed from tail
<b>\${var:#pat}</b>	<i>var</i> unless <i>pat</i> matches, then empty
<b>\${var/p/r}</b>	One occurrence of <i>p</i> replaced by <i>r</i>
<b>\${var//p/r}</b>	All occurrences of <i>p</i> replaced by <i>r</i>
<b>\${#var}</b>	Length of <i>var</i> in words (array) or bytes
<b>\${^var}</b>	Expand elements like brace expansion
<b>\${=var}</b>	Split words of result like lesser shells
<b>\${~var}</b>	Allow globbing, file expansion on result
<b>\${\${var%p}%#q}</b>	Apply <i>%p</i> then <i>%q</i> to <i>var</i>

Parameter flags in parentheses, immediately after left brace:

<b>%</b>	Expand %s in result as in prompts
<b>@</b>	Array expand even in double quotes
<b>A</b>	Create array parameter with <b>\${...=...}</b>
<b>a</b>	Array index order, so <b>0a</b> is reversed
<b>c</b>	Count characters for <b>\${#var}</b>
<b>C</b>	Capitalize result
<b>e</b>	Do parameter, comand, arith expansion
<b>f</b>	Split result to array on newlines
<b>F</b>	Join arrays with newlines between elements
<b>i</b>	<b>oi</b> or <b>Oi</b> sort case independently
<b>k</b>	For associative array, result is keys
<b>L</b>	Lower case result
<b>n</b>	<b>on</b> or <b>On</b> sort numerically
<b>o</b>	Sort into ascending order
<b>O</b>	Sort into descending order
<b>P</b>	Interpret result as parameter name, get value

<b>q</b>	Quote result with backslashes
<b>qq</b>	Quote result with single quotes
<b>qqq</b>	Quote result with double quotes
<b>qqqq</b>	Quote result with <b>'...'</b>
<b>Q</b>	Strip quotes from result
<b>t</b>	Output type of variable (see below)
<b>u</b>	Unique: remove duplicates after first
<b>U</b>	Upper case result
<b>v</b>	Include value in result; may have <b>(kv)</b>
<b>V</b>	Visible representation of special chars
<b>w</b>	Count words with <b>\${#var}</b>
<b>W</b>	Same but empty words count
<b>X</b>	Report parsing errors (normally ignored)
<b>z</b>	Split to words using shell grammar
<b>p</b>	Following forms recognize print \-escapes
<b>j:str:</b>	Join words with <i>str</i> between
<b>l:x:</b>	Pad with spaces on left to width <i>x</i>
<b>l:x::s1:</b>	Same but pad with repeated <i>s1</i>
<b>l:x::s1::s2:</b>	Same but <i>s2</i> used once before any <i>s1</i> s
<b>r:x::s1::s2:</b>	Pad on right, otherwise same as <b>l</b> forms
<b>s:str:</b>	Split to array on occurrences of <i>str</i>
<b>S</b>	With patterns, search substrings
<b>I:exp:</b>	With patterns, match <i>exp</i> th occurrence
<b>B</b>	With patterns, include match beginning
<b>E</b>	With patterns, include match end
<b>M</b>	With patterns, include matched portion
<b>N</b>	With patterns, include length of match
<b>R</b>	With patterns, include unmatched part (rest)

Delimiters shown as **:str:** may be any pair of chars or matched parentheses (*str*), {*str*}, [*str*], <*str*>.

Order of rules:

1. Nested substitution: from inside out
2. Subscripts: **\${arr[3]}** extract word; **\${str[2]}** extract character; **\${arr[2,4]}**, **\${str[4,8]}** extract range; **-1** is last word/char, **-2** previous etc.
3. **\${(P)var}** replaces name with value
4. **"\$array"** joins array, may use **(j:str:)**
5. Nested subscript e.g. **\${\${var[2,4]}[1]}**
6. **#**, **%**, **/** etc. modifications
7. Join if not joined and **(j:str:)**, **(F)**
8. Split if **(s)**, **(z)**, **(z)**, **=**
9. Split if **SH\_WORD\_SPLIT**
10. Apply **(u)**
11. Apply **(o)**, **(O)**
12. Apply **(e)**
13. Apply **(l.str.)**, **(r.str.)**
14. If single word needed for context, join with **\$IFS[1]**.

Types shown with **(t)** have basic type **scalar**, **array**, **integer**, **float**, **assocation**, then hyphen-separated words from following list:

<b>local</b>	Parameter is local to function
<b>left</b>	Left justified with <b>typeset -L</b>
<b>right_blanks</b>	Right justified with <b>typeset -R</b>
<b>right_zeros</b>	Right justified with <b>typeset -Z</b>
<b>lower</b>	Lower case forced with <b>typeset -l</b>
<b>upper</b>	Upper case forced with <b>typeset -u</b>
<b>readonly</b>	Read-only, <b>typeset -r</b> or <b>readonly</b>
<b>tag</b>	Tagged as <b>typeset -t</b> (no special effect)
<b>export</b>	Exported with <b>export</b> , <b>typeset -x</b>
<b>unique</b>	Elements unique with <b>typeset -U</b>
<b>hide</b>	Variable not special in func ( <b>typeset -h</b> )
<b>hideval</b>	<b>typeset</b> hides value ( <b>typeset -H</b> )
<b>special</b>	Variable special to shell

**Parameters (Variables)**

Parameters set by shell, † denotes special to shell (may not be reused except by hiding with typeset -h in functions)

†!	Process ID of last background process
†#	Number of arguments to script or function
†ARGC	Same
†\$	Process ID of main shell process
†-	String of single letter options set
†*	Positional parameters
†argv	Same
†@	Same, but does splitting in double quotes
†?	Status of last command
†0	Name of shell, usually reflects functions
†_	Last argument of previous command
CPUTYPE	Machine type (run time)
†EGID	Effective GID (via system call), set if root
†EUID	Effective UID (via system call), set if root
†ERRNO	Last system error number
†GID	Real group ID (via system call), set if root
HISTCMD	The current history line number
HOST	The host name
†LINENO	Line number in shell, function
LOGNAME	Login name (exported by default)
MACHTYPE	Machine type (compile time)
OLDPWD	Previous directory
†OPTARG	Argument for option handled by getopt
†OPTIND	Index of positional parameter in getopt
OSTYPE	Operating system type (compile time)
†pipestatus	Array giving statuses of last pipeline
†PPID	Process ID of parent of main shell
PWD	Current directory
†RANDOM	A pseudo-random number, repeating
†SECONDS	Seconds since shell started
†SHLVL	Depth of current shell
signals	Array giving names of signals
†status	Status of last command
†TRY_BLOCK_ERROR	In always block, 1 if error in try block
TTY	Terminal associated with shell if any
†TTYIDLE	Time for which terminal has been idle
†UID	Real user ID (via system call), set if root

†USERNAME	Name for \$UID, set if root
VENDOR	Operating system vendor (compile time)
ZSH_NAME	Base name of command used to start shell
ZSH_VERSION	Version number of shell

Parameters used by the shell if set: : indicates arrays with corresponding colon-separated paths e.g. **cdpath** and **CDPATH**:

ARGVO	Export to set name of external command
BAUD	Baud rate: compensation for slow terminals
†cdpath :	Directories searched for <b>cd</b> target
†COLUMNS	Width of screen
DIRSTACKSIZE	Maximum size of stack for <b>pushd</b>
ENV	File to source when started as <b>sh</b> or <b>ksh</b>
FCEDIT	Default editor used by <b>fc</b>
†fignore :	List of suffixes ignored in file completion
†fpath :	Directories to search for autoloading
†histchars	History, quick replace, comment chars
†HISTCHARS	Same, deprecated
HISTFILE	File for reading and writing shell history
†HISTSIZE	Number of history lines kept internally
†HOME	Home directory for ~ and default <b>cd</b> target
†IFS	Characters that separate fields in words
KEYTIMEOUT	Time to wait for rest of key seq (1/100 s)
†LANG	Locale (usual variable, <b>LC_*</b> override)
†LC_ALL	Locale (overrides <b>LANG</b> , <b>LC_*</b> )
†LC_COLLATE	Locale for sorting etc.
†LC_CTYPE	Locale for character handling
†LC_MESSAGES	Locale for messages
†LC_NUMERIC	Locale for decimal point, thousands
†LC_TIME	Locale for date and time
†LINES	Height of screen
LISTMAX	Number of completions shown w/o asking
LOGCHECK	Interval for checking <b>\$watch</b>
MAIL	Mail file to check ( <b>\$mailpath</b> overrides)
MAILCHECK	Mail check interval, secs (before prompt)
†mailpath :	List of files to check for new mail
†manpath :	Directories to find manual, used by <b>man</b>
†module_path :	Directories for <b>zmodload</b> to find modules
†NULLCMD	Command used if only redirection given
†path :	Command search path
†POSTEDIT	Termcap strings sent to terminal after edit

†PS1, PROMPT, prompt	Printed at start of first line of output; see above for escape sequences for all <b>PS</b> s
†PS2, PROMPT2	Printed for continuation lines
†PS3, PROMPT3	Print within <b>select</b> loop
†PS4, PROMPT4	For tracing execution ( <b>xtrace</b> option)
†psvar :	Used with <b>%nv</b> in prompts
†READNULLCMD	Command used when only input redir given
REPORTTIME	Show report if command takes this long (s)
REPLY	Used to return a value e.g. by <b>read</b>
reply	Used to return array value
†RPS1, RPROMPT	Printed on right of screen for first line
†RPS2, RPROMPT2	Printed on right of screen for continuation line
SAVEHIST	Max number of history lines saved
†SPROMPT	Prompt when correcting spelling
STTY	Export with <b>stty</b> arguments to command
†TERM	Type of terminal in use ( <b>xterm</b> etc.)
TIMEFMT	Format for reporting usage with <b>time</b>
TMOUT	Send <b>SIGALRM</b> after seconds of inactivity
TMPPREFIX	Path prefix for shell's temporary files
†watch :	List of users or <b>all</b> , <b>notme</b> to watch for
WATCHFMT	Format of reports for <b>\$watch</b>
†WORDCHARS	Chars considered parts of word by <b>zle</b>
ZBEEP	String to replace beeps in line editor
ZDOTDIR	Used for startup files instead of ~ if set



**Tests and numeric expressions**

Usually used after if, while, until or with && or ||, but the status may be useful anywhere e.g. as implicit return status for function.

File tests, e.g. `[[ -e file ]]`:

<b>-a</b>	True if <i>file</i> exists
<b>-b</b>	True if <i>file</i> is block special
<b>-c</b>	True if <i>file</i> is character special
<b>-d</b>	True if <i>file</i> is directory
<b>-e</b>	True if <i>file</i> exists
<b>-f</b>	True if <i>file</i> is a regular file (not special or directory)
<b>-g</b>	True if <i>file</i> has setgid bit set (mode includes 02000)
<b>-h</b>	True if <i>file</i> is symbolic link
<b>-k</b>	True if <i>file</i> has sticky bit set (mode includes 02000)
<b>-p</b>	True if <i>file</i> is named pipe (FIFO)
<b>-r</b>	True if <i>file</i> is readable by current process
<b>-s</b>	True if <i>file</i> has non-zero size
<b>-u</b>	True if <i>file</i> has setuid bit set (mode includes 04000)
<b>-w</b>	True if <i>file</i> is writeable by current process
<b>-x</b>	True if <i>file</i> executable by current process
<b>-L</b>	True if <i>file</i> is symbolic link
<b>-O</b>	True if <i>file</i> owned by effective UID of current process
<b>-G</b>	True if <i>file</i> has effective GID of current process
<b>-S</b>	True if <i>file</i> is a socket (special communication file)
<b>-N</b>	True if <i>file</i> has access time no newer than mod time

Other single argument tests, e.g. `[[ -n str ]]`:

<b>-n</b>	True if <i>str</i> has non-zero length
<b>-o</b>	True if option <i>str</i> is set
<b>-t</b>	True if <i>str</i> (number) is open file descriptor
<b>-z</b>	True if <i>str</i> has zero length

Multiple argument tests e.g. `[[ a -eq b ]]`: numerical expressions may be quoted formulae e.g. `'1*2'`:

<b>-nt</b>	True if file <i>a</i> is newer than file <i>b</i>
<b>-ot</b>	True if file <i>a</i> is older than file <i>b</i>
<b>-ef</b>	True if <i>a</i> and <i>b</i> refer to same file (i.e. are linked)
<b>=</b>	True if string <i>a</i> matches pattern <i>b</i>
<b>==</b>	Same but more modern (and still not often used)

<b>!=</b>	True if string <i>a</i> does not match pattern <i>b</i>
<b>&lt;</b>	True if string <i>a</i> sorts before string <i>b</i>
<b>&gt;</b>	True if string <i>a</i> sorts after string <i>b</i>
<b>-eq</b>	True if numerical expressions <i>a</i> and <i>b</i> are equal
<b>-ne</b>	True if numerical expressions <i>a</i> and <i>b</i> are not equal
<b>-lt</b>	True if <i>a</i> < <i>b</i> numerically
<b>-gt</b>	True if <i>a</i> > <i>b</i> numerically
<b>-le</b>	True if <i>a</i> ≤ <i>b</i> numerically
<b>-ge</b>	True if <i>a</i> ≥ <i>b</i> numerically

Combining expressions: *expr* is any of the above, or the result of any combination of the following:

<b>( expr )</b>	Group tests
<b>! expr</b>	True if <i>expr</i> is false and vice versa
<b>exprA &amp;&amp; exprB</b>	True if both expressions true
<b>exprA    exprB</b>	True if either expression true

For complicated numeric tests use `(( expr ))` where *expr* is a numeric expression: status is 1 if *expr* is non-zero else 0. Same syntax used in `$( ( expr ) )` substitution. Precedences of operators from highest to lowest are:

- *func(arg...)*, numeric constant (e.g. **3**, **-4**, **3.24**, **-14.6e-10**), *var* (does not require \$ in front unless some substitution e.g. `${#var}` is needed, \$ is error if *var* is to be modified)
- **( expr )**
- **!**, **~**, **++** (post- or preincrement), **--** (post- or predecrement), unary **+**, unary **-**
- **&**
- **^**
- **|**
- **\*\*** (exponentiation)
- **\***, **/**, **%**
- binary **+**, binary **-**
- **<<**, **>>**
- **<**, **<=**, **>**, **>=**
- **==**, **!=**
- **&&**
- **||**, **^^**
- **?** (ternary operator)

- **:** (true/false separator for ternary operator)
- **=**, **+=**, **-=**, **\*=**, **/=**, **%=**, **\*\*=**, **&=**, **^=**, **|=**, **<<=**, **>>=**, **&&=**, **^^=**, **||=**
- **,** (as in C, evaluate both sides and return right hand side).

For functions use **zmodload -i zsh/mathfunc**; functions available are as described in C math library manual:

- Single floating point argument, return floating point: **acos**, **acosh**, **asin**, **asinh**, **atan** (optional second argument like C `atan2`), **atanh**, **cbirt**, **ceil**, **cos**, **cosh**, **erf**, **erfc**, **exp**, **expm1**, **fabs**, **floor**, **gamma**, **j0**, **j1**, **lgamma**, **log**, **log10**, **log1p**, **logb**, **sin**, **sinh**, **sqrt**, **tan**, **tanh**, **y0**, **y1**
- Single floating point argument, return integer: **ilogb**
- No arguments, return integer: **signgam** (remember parentheses)
- Two floating point arguments, return floating point: **copysign**, **fmod**, **hypot**, **nextafter**
- One integer, one floating point argument, return floating point: **jn**, **yn**
- One floating point, one integer argument, return floating point: **ldexp**, **scalb**
- Either integer or floating point, return same type: **abs**
- Coerce to floating point: **float**
- Coerce to integer: **int**
- Optional string argument (read/write variable name), return floating point: **rand48**

Example use:

```
zmodload -i zsh/mathfunc
float x
(( x = 26.4 * sqrt(2) ))
print $(( log(x)/2 ))
```

<b>Completion</b>			
Load new completion system with:			
<b>autoload -Uz compinit</b>			
<b>compinit</b>			
Configuration: uses styles			
<b>zstyle context style value...</b>			
where context may be a pattern matching the following form:			
<b>:completion:func:completer:cmd:arg:tag</b>			
in which:			
<b>completion</b>			
Literal string always used by completion functions			
<b>func</b>			
Name of directly called widget, blank for contextual completion			
<b>completer</b>			
Method of completion e.g. <b>complete</b> ; see below			
<b>cmd</b>			
Name of command being completed, or special command context			
<b>arg</b>			
Only valid with standard parsing: <b>arg-n</b> for <i>n</i> th argument			
<b>option-opt-n</b> for <i>n</i> th argument of option opt			
<b>tag</b>			
Indication of type of thing to be completed at this point.			
Completers († indicates modifiers existing or later completions):			
<b>†_all_matches</b>	Later completers add all matches		
<b>_approximate</b>	Complete with errors in part so far		
<b>_complete</b>	Basic completion		
<b>_correct</b>	Correct word already typed		
<b>_expand</b>	Perform shell expansions		
<b>_expand_alias</b>	Expand aliases only		
<b>_history</b>	Complete words from shell history		
<b>†_ignored</b>	Reinstate matches omitted		
<b>†_list</b>	List on first completion, insert on second		
<b>_match</b>	Complete using patterns from line		
<b>†_menu</b>	Menu completion, no menu selection		
<b>†_oldlist</b>	Use existing list before generating new one		
<b>_prefix</b>	Complete ignoring what's after cursor		
Command contexts: any command name plus the special contexts:			
<b>-array-value-</b>	Element in array		
<b>-brace-parameter-</b>	Parameter within <b>\${...}</b>		
<b>-assign-parameter-</b>	Left hand side of assignment		
<b>-command-</b>	Word in command position		
<b>-condition-</b>	Word in <b>[[ ... ]]</b> condition		
<b>-default-</b>	Word with no specific completion		
<b>-equal-</b>	Word beginning with equals sign		
<b>-first-</b>	Tried first, may set <b>_compskip</b>		
<b>-math-</b>	Inside arithmetic such as <b>(( ... ))</b>		
<b>-parameter-</b>	Parameter with bare <b>\$</b> in front		
<b>-redirect-</b>	Word after redirection operator		
<b>-subscript-</b>	Inside parameter subscript		
<b>-tilde-</b>	Between <b>~</b> and first <b>/</b> of argument		
<b>-value-</b>	Right hand side of assignment		
Tags:			
<b>accounts</b>	For users-hosts style		
<b>all-expansions</b>	When expanding, everything at once		
<b>all-files</b>	All files rather than a subset		
<b>arguments</b>	Command arguments		
<b>arrays</b>	Names of array parameters		
<b>association-keys</b>	Keys of associative arrays		
<b>bookmarks</b>	Bookmarks for URLs, ZFTP, etc.		
<b>builtins</b>	Names of builtin commands		
<b>characters</b>	Character classes, stty characters		
<b>colormapids</b>	X colormap IDs		
<b>colors</b>	Names of colors, usually X		
<b>commands</b>	External commands, subcommands		
<b>contexts</b>	Contexts in zstyle		
<b>corrections</b>	Possible approximations, corrections		
<b>cursors</b>	X cursor names		
<b>default</b>	Nothing specific in certain contexts		
<b>descriptions</b>	Used in format style for matches		
<b>devices</b>	Device special files		
<b>directories</b>	Directories		
<b>directory-stack</b>	Entries in pushd directory stack		
<b>displays</b>	X displays		
<b>domains</b>	Network domain (DNS) names		
<b>expansions</b>	Individual expansions instead of all		
<b>file-descriptors</b>	Numbers of open file descriptors		
<b>files</b>	Generic file matching tag		
<b>fonts</b>	X font names		
<b>fstypes</b>	Files system types for mount etc.		
<b>functions</b>	Shell functions, possibly other types		
<b>globbed-files</b>	Names of files matched by pattern		
<b>groups</b>	UNIX groups		
<b>history-words</b>	Words from shell history		
<b>hosts</b>	Names of network hosts		
<b>indexes</b>	Indexes of arrays		
<b>jobs</b>	Shell jobs		
<b>interfaces</b>	Network interfaces (as from ifconfig)		
<b>keymaps</b>	ZLE keymaps		
<b>keysyms</b>	Names of X keysyms		
<b>libraries</b>	Names of system libraries		
<b>limits</b>	System resource limits		
<b>local-directories</b>	Subdirectories of current directories		
<b>manuals</b>	Names of manual pages		
<b>mailboxes</b>	E-mail folders		
<b>maps</b>	NIS maps etc.		
<b>messages</b>	Used in format style for messages		
<b>modifiers</b>	X modifiers		
<b>modules</b>	Shell modules etc.		
<b>my-accounts</b>	Own accounts, with users-hosts style		
<b>named-directories</b>	Directories named by a parameter		
<b>names</b>	Names of all sorts		
<b>newsgroups</b>	USENET newgroups		
<b>nicknames</b>	Nicknames of NIS maps		
<b>options</b>	Options to commands		
<b>original</b>	Original when correcting, expanding		
<b>other-accounts</b>	Other accounts with users-hosts style		
Tags continued:			
<b>packages</b>	RPM, Debian etc. packages		
<b>parameters</b>	Names of shell parameters		
<b>path-directories</b>	Directories under \$cdpath		
<b>paths</b>	Used with assorted directory paths		
<b>pods</b>	Perl documentation		
<b>ports</b>	TCP, UDP protos		
<b>prefixes</b>	URL etc. prefixes		
<b>printers</b>	Names of print queues		
<b>processes</b>	PIDs		
<b>processes-names</b>	Names of processes in killall		

<b>sequences</b>	MH sequences etc.
<b>sessions</b>	ZFTP sessions etc.
<b>signals</b>	System signal names, HUP etc.
<b>strings</b>	Assorted strings, e.g. second arg of cd
<b>styles</b>	Styles in zstyle
<b>suffixes</b>	Filename extensions
<b>tags</b>	Tags used with rpm etc.
<b>targets</b>	Targets inside Makefiles
<b>time-zones</b>	Time zones with TZ parameter etc.
<b>types</b>	Assorted types of anything
<b>urls</b>	Used with web addresses
<b>users</b>	Names of users
<b>values</b>	Values in lists
<b>variant</b>	Used when picking variant of command
<b>visuals</b>	X visuals
<b>warnings</b>	Used in the format style for warnings
<b>widgets</b>	Names of zsh widgets
<b>windows</b>	IDs of X windows
<b>zsh-options</b>	Shell options

Styles († indicates on by default):

<b>accept-exact</b>	Accept exact match even if ambiguous
<b>†add-space</b>	Add a space after expansions
<b>ambiguous</b>	Cursor after ambiguous path component
<b>assign-list</b>	<b>PATH</b> -style list on assignment
<b>auto-description</b>	String for option desc's without specific
<b>avoid-completer</b>	Avoid completer with <u><b>_all_matches</b></u>
<b>cache-path</b>	Path to top of various caches
<b>cache-policy</b>	Function to decide on cache rebuilding
<b>call-command</b>	If true, use external (slow) command
<b>command</b>	External command to call (+args)
<b>command-path</b>	Override <b>PATH</b> for commands to match
<b>commands</b>	Default sys init commands (start etc.)
<b>complete</b>	Complete aliases ( <u><b>_expand_alias</b></u> )
<b>completer</b>	The list of completers to try (see above)
<b>†condition</b>	Delay insertion of matches ( <u><b>_list</b></u> )
<b>disabled</b>	Disabled aliases ( <u><b>_expand_alias</b></u> )
<b>disable-stat</b>	If set, <b>_cvs</b> uses ls instead of zsh/stat
<b>domains</b>	Net domains ( <b>/etc/resolv.conf</b> )
<b>expand</b>	For <b>prefix</b> , <b>suffix</b> in multiple parts
<b>fake</b>	Add <b>value:desc</b> fake completions

<b>fake-files</b>	<i>dir:names</i> add <i>names</i> in <i>dir</i>
<b>fake-parameters</b>	Params to complete even if not yet set
<b>file-patterns</b>	<i>pattern:tag</i> generates files with tag
<b>file-sort</b>	size, links, time, access, inode, reverse
<b>filter</b>	In LDAP, attributes for filtering
<b>force-list</b>	Just list matches: <b>always</b> or number
<b>format</b>	Desc string, <b>%d</b> shows specific desc
<b>†glob</b>	Attempt glob expansion ( <u><b>_expand</b></u> )
<b>†global</b>	Global aliases ( <u><b>_expand_alias</b></u> )
<b>group-name</b>	Name groups shown together by tag
<b>group-order</b>	Order groups shown together by tag
<b>groups</b>	Unix groups, as per <b>/etc/group</b>
<b>hidden</b>	Complete but don't list matches
<b>hosts</b>	List of host names, as <b>/etc/hosts</b>
<b>hosts-ports</b>	List of <i>hosts:ports</i> for TCP/UDP
<b>ignore-line</b>	Don't complete words already present
<b>ignore-parents</b>	<b>parent</b> or <b>pwd</b> : ignore parent dirs
<b>ignored-patterns</b>	If pattern matched, don't complete
<b>insert</b>	All matches at once ( <u><b>_all_matches</b></u> )
<b>insert-ids</b>	Convert <b>%cmd</b> to unambiguous PID
<b>insert-tab</b>	Insert TAB if no non-whitespace yet
<b>insert-unambiguous</b>	Only menu complete when no prefix to insert
<b>keep-prefix</b>	Try to keep expandable prefix
<b>last-prompt</b>	Return to last editing line if possible
<b>list</b>	Control listing when history completing
<b>list-colors</b>	Color specs like <b>LS_COLORS</b>
<b>†list-grouped</b>	Grouped listing shown more compactly
<b>list-packed</b>	All matches shown more compactly
<b>list-prompt</b>	Prompt when scrolling completions
<b>list-rows-first</b>	Increment rows first in lists
<b>list-suffixes</b>	Show ambiguous bits of multiple paths
<b>list-separator</b>	Separates description in verbose list
<b>local</b>	<i>host:path:dir</i> for URLs as files
<b>mail-directory</b>	Directory for mailbox files ( <b>~/Mail</b> )
<b>match-original</b>	Add * when matching ( <u><b>_match</b></u> )
<b>matcher</b>	Apply match control syntax per tag
<b>matcher-list</b>	Apply match control syntax globally
<b>max-errors</b>	Max errors allowed in approx/correct
<b>max-matches-width</b>	Cols to reserve for matches (not desc)
<b>menu</b>	Use menu completion

<b>muttrc</b>	Alternative for <b>~/muttrc</b>
<b>numbers</b>	Prefer job numbers instead of name
<b>old-list</b>	Retain list of matches ( <u><b>_oldlist</b></u> )
<b>old-matches</b>	Use old match list ( <u><b>_all_matches</b></u> )
<b>old-menu</b>	Keep list for meus ( <u><b>_oldlist</b></u> )
<b>original</b>	Add original match for approx/correct
<b>packageset</b>	For arguments of Debian <b>dpkg</b>
<b>path</b>	For X colors, path to <b>rgb.txt</b>
<b>pine-directory</b>	Directory for PINE mailboxes
<b>ports</b>	TCP/IP services ( <b>/etc/services</b> )
<b>prefix-hidden</b>	Hide common prefix e.g. in options
<b>prefix-needed</b>	Common prefix must be typed by user
<b>preserve-prefix</b>	Initial file patterns to leave alone
<b>range</b>	Range of words in history to consider
<b>†regular</b>	Complete regular aliases

Styles continued:

<b>†remote-access</b>	Control remote access for e.g. <b>_cvs</b>
<b>remove-all-dups</b>	Never complete duplicates in history
<b>select-prompt</b>	Prompt shown in menu selection
<b>select-scroll</b>	Lines to scroll in menu selection
<b>separate-sections</b>	Manual sections used as part of tag
<b>show-completer</b>	Show progress of completers as msg
<b>single-ignored</b>	Control <b>_ignore</b> when single match
<b>sort</b>	Override sorting of matches
<b>special-dirs</b>	Add <b>.</b> and <b>..</b> to file list
<b>squeeze-slashes</b>	<b>fo//ba</b> is <b>fo/ba</b> not <b>fo/*/ba</b>
<b>stop</b>	Pause before looping shell history
<b>strip-comments</b>	Remove display name from email addr
<b>subst-globs-only</b>	Only take expansions from globbing
<b>†substitute</b>	When expanding, first try subst
<b>†suffix</b>	Only expand path with no <b>/suffix</b>
<b>tag-order</b>	Preference order for tags in context
<b>urls</b>	Determine where URLs are taken from
<b>use-cache</b>	Control caching for various commands
<b>use-compctl</b>	Use <b>compctl</b> -style completions
<b>use-perl</b>	Use simpler Perl code for <b>_make</b>
<b>users</b>	List of user names
<b>users-hosts</b>	List of <i>user@host</i> possibilities
<b>users-hosts-ports</b>	List of <i>user@host:port</i>
<b>†verbose</b>	Verbose output e.g. option descriptions
<b>word</b>	Line changes based on current word

Using **\_arguments** for parsing standard command arguments:  
Three arguments give argument/option selector, message to output, action to take. Examples:

```
1:msg:_comp      First arg; show msg, exec _comp
1::msg:_comp     Same for optional argument
:msg:_comp       Arg number inferred from position
*:msg:_comp      Any of the remaining args ("rest args")
*:::msg:_comp    words etc. set to normal args
*::::msg:_comp   ... set to args for this chunk
-foo            Complete option -foo
+foo            Complete option +foo
-+foo           Complete -foo or +foo
*-foo           Option may occur multiple times
-foo:esg:_comp   Option has arg in same word
-foo+:msg:_comp  Option has arg in same or next word
-foo=:msg:_comp  Option arg -foo=bar or -foo bar
-foo=:msg:_comp  Option arg is -foo=bar only
-foo[desc]       Option has description desc
*:pat:msg:_comp  Complete words up to pat
*:pat:msg:_comp  Modify words etc. for args
(-goo -boo)-foo  -foo excludes -goo, -boo
(*)-foo          -foo excludes rest args as matches
(:)-foo          -foo excludes normal args
(-)-foo          -foo excludes all options
!-foo            -foo should not be completed
*:msg:<space>     Show message but don't complete
*:msg:(a b)       Matches are listed items
*:msg:((a\dsc))   Matches with descriptions
*:msg:->string     Array state has string if matched
*:msg:{code}       Shell code generates matches
*:msg:= action     Insert dummy argument first
*:msg:_comp arg    Call _comp with additional args
*:msg: _comp arg   Call _comp with only given arg
-a - set1 -c - ... Common and specific completion sets
- "(set1)" -c - ... Mutually exclusive sets
-s                Allow combined single letters
-sw              Same, even if option has args
--              Guess options by using --help
-- -i pat        Same, ignoring options matching pat
```

Examples of other utility functions:

```
_alternative \
'users:user:_users' \
'hosts:host:_hosts'
```

Either users or hosts (tag, description, action)

```
_describe setdesc arr1 --
```

Associate descriptions with completions; *arr1* contains *completion:description* entries

```
_message text-msg
```

Don't complete, just output *text-msg*

```
_multi_parts sep array
```

Complete by parts with separator *sep*, *\$array* contains full matches.

```
_path_files
```

Complete files including partial paths; **\_files** is smart front end; options **-f** all files (default), **-g pat** matching *pat* (with **\_files** maybe directories too), **-/** directories only, **-W dirs** paths in which files are found, **-F files** files to ignore, overrides **ignored-patterns**

```
_sep_parts arr1 sep1 arr2 sep2 .....
```

Elements from *arr1*, then separator, then elements from *arr2*, etc.

```
_values -s sep desc spec1 spec2 ...
```

Complete multiple values separated by *sep*; values are given by *specs*, each of which is similar to **\_arguments** option *spec* without leading -

```
_wanted thing expl 'my things' \
compadd mything1 mything2 ...
```

Typical way of adding completions **mything1** etc. with tag **things** and description **my things**; **expl** should be local variable. Use single tag, c.f. **\_tags** and **\_requested**

```
_tags tag1 tag2
```

```
_requested tag
```

Implement loops over different tags

```
_all_labels tag expl descr compcommand
```

```
_next_label tag expl descr
```

Implement loops over different labels for each **\_requested** tag

Zsh line editor (zle)											
Builtin widgets, emacs binding, vicmd binding, viins binding; € denotes escape key:											
accept-and-hold	€a			execute-last-named-cmd	€z			redisplay	^R	^R	
accept-and-infer-next-history				execute-name-cmd	€x			redo			
accept-and-menu-complete				expand-cmd-path				reset-prompt			
accept-line	^M	^M	^M	expand-history	€!			reverse-menu-complete			
accept-line-and-down-history	^O			expand-or-complete	^I		^I	run-help	€h		
argument-base				expand-or-complete-prefix				self-insert	...		...
backward-char	^B			expand-word	^X*			self-insert-unmeta	€^M		
backward-delete-char	^H			forward-char			^F	send-break	^G		
backward-delete-word				forward-word	€f			set-mark-command	^@		
backward-kill-line				get-line	€g			spell-word	€s		
backward-kill-word	^W			gosmacs-transpose-chars				set-local-history			
backward-word	€b			history-beginning-search-backward				transpose-chars	^T		
beep				history-beginning-search-forward				transpose-words	€t		
beginning-of-buffer-or-history	€<			history-incremental-search-backward	^R			undefined-key			
beginning-of-history				history-incremental-search-forward	^Xr			undo	^_		
beginning-of-line	^A			history-search-backward	^S			universal-argument	€u		
beginning-of-line-hist				history-search-forward	^Xs			up-case-word		^p	
capitalize-word	€c			infer-next-history	€p			up-history		k	up
clear-screen	^L	^L	^L	insert-last-word	€n			up-line-or-history	^p		
complete-word				kill-buffer	^x^n			up-line-or-search			
copy-prev-word	€^_			kill-line	€_			vi-add-eol		A	
copy-prev-shell-word				kill-region	^X^K			vi-add-next		a	
copy-region-as-kill	€w			kill-whole-line	^K			vi-backward-blank-word		B	
delete-char	^D			list-choices	^U			vi-backward-char		h	^H left
delete-char-or-list				list-expand	€d	^d	^d	vi-backward-delete-char		X	^H
delete-word				magic-space	€Xg	^G	^G	vi-backward-kill-word			^W
describe-key-briefly				menu-complete				vi-backward-word		b	
digit-argument	€0..1..9			menu-expand-or-complete				vi-beginning-of-line			
down-case-word	€l			neg-argument	€-			vi-caps-lock-panic			
down-history		^n		overwrite-mode	^X^O			vi-change		c	
down-line-or-history	^n	j	down	pound-insert				vi-change-eol		C	
down-line-or-search				push-input				vi-change-whole-line		S	
emacs-backward-word				push-line				vi-cmd-mode	^XV		€
emacs-forward-word				push-line-or-edit	€q			vi-delete		d	
end-of-buffer-or-history				quoted-insert				vi-delete-char		x	
end-of-history	€>			quote-line				vi-digit-or-beginning-of-line		0	
end-of-line	^E			quote-region	€"			vi-down-line-or-history		+	
end-of-line-hist				recursive-edit				Builtin widgets cont.:			
end-of-list								vi-end-of-line		\$	
exchange-point-and-mark	^X^X							vi-fetch-history		G	
								vi-find-next-char	^X^F	f	
								vi-find-next-char-skip		t	
								vi-find-prev-char		F	

<b>vi-find-prev-char-skip</b>	<b>T</b>		<b>what-cursor-position</b>	<b>^X=</b>	<b>\Mx, \M-x</b>	Set 8 <sup>th</sup> bit in character
<b>vi-first-non-blank</b>	<b>^</b>		<b>where-is</b>		<b>\Cx, \C-x</b>	Control character e.g. <b>\C-a</b>
<b>vi-forward-blank-word</b>	<b>W</b>		<b>which-command</b>	<b>€?</b>	<b>^x</b>	Control character e.g. <b>^a</b> (same as <b>^A</b> )
<b>vi-forward-blank-word-end</b>	<b>E</b>		<b>yank</b>	<b>^y</b>	<b>^?</b>	Delete
<b>vi-forward-char</b>	<b>l</b>	<i>right</i>	<b>yank-pop</b>	<b>€y</b>	<b>\\</b>	Single backslash
<b>vi-forward-word</b>	<b>w</b>					
<b>vi-forward-word-end</b>	<b>e</b>		Special parameters inside user-defined widgets; † indicates readonly:			
<b>vi-goto-column</b>	<b>€ </b>	<b> </b>	<b>BUFFER</b>	Entire editing buffer	<b>emacs</b>	Like Emacs editor
<b>vi-goto-mark</b>		<b>.</b>	<b>BUFFERLINES</b>	Number of screen lines for full buffer	<b>viins</b>	Like Vi editor in insert mode
<b>vi-goto-mark-line</b>		<b>.</b>	<b>†CONTEXT</b>	<b>start, cont, select, vared</b>	<b>vicmd</b>	Like Vi editor in command mode
<b>vi-history-search-backward</b>	<b>/</b>		<b>CURSOR</b>	Index of cursor position into <b>\$BUFFER</b>	<b>.safe</b>	Emergency keymap, not modifiable
<b>vi-history-search-forward</b>	<b>?</b>		<b>CUTBUFFER</b>	Last item to be killed		
<b>vi-indent</b>	<b>&gt;</b>		<b>HISTNO</b>	Currently history line being retrieved	Examples of key binding:	
<b>vi-insert</b>	<b>i</b>		<b>†KEYMAP</b>	Currently selected keymap	<b>bindkey '^xt' gosmacs-transpose-chars</b>	
<b>vi-insert-bol</b>	<b>I</b>		<b>†KEYS</b>	Keys typed to invoke current widget	<b>bindkey '\e[2~' overwrite-mode</b>	
<b>vi-join</b>	<b>^X^J</b>		<b>killring</b>	Array of previously killed items, can resize	<b>bindkey -M viins '^u' backward-kill-line</b>	
<b>vi-kill-eol</b>	<b>D</b>		<b>†LASTSEARCH</b>	Last search string in interactive search	<b>bindkey -s '^x^z' '\eqsuspend\n'</b>	
<b>vi-kill-line</b>		<b>^U</b>	<b>†LASTWIDGET</b>	Last widget to be executed	<b>autoload -Uz replace-string</b>	
<b>vi-match-bracket</b>	<b>^X^B%</b>		<b>LBUFFER</b>	Part of buffer left of cursor	<b>zle -N replace-string</b>	
<b>vi-open-line-above</b>	<b>O</b>		<b>MARK</b>	Index of mark position into <b>\$BUFFER</b>	<b>bindkey '\er' replace-string</b>	
<b>vi-open-line-below</b>	<b>o</b>		<b>NUMERIC</b>	Numeric argument passed with widget	<b>zle -N replace-string replace-pattern</b>	
<b>vi-oper-swap-case</b>			<b>†PENDING</b>	Number of bytes still to be read	<b>bindkey '\e%' replace-pattern</b>	
<b>vi-pound-insert</b>			<b>†PREBUFFER</b>	Input already read (no longer being edited)	See <b>man zshcontrib</b> for supplied editing functions such as	
<b>vi-put-after</b>	<b>P</b>		<b>PREDISPLAY</b>	Text to display before editable buffer	<b>replace-string.</b>	
<b>vi-put-before</b>	<b>p</b>		<b>POSTDISPLAY</b>	Text to display after editable buffer		
<b>vi-quoted-insert</b>		<b>^V</b>	<b>RBUFFER</b>	Part of buffer starting from cursor		
<b>vi-repeat-change</b>	<b>.</b>		<b>WIDGET</b>	Name of widget being executed		
<b>vi-repeat-find</b>	<b>;</b>		<b>WIDGETFUNC</b>	Name of function implementing <b>\$WIDGET</b>		
<b>vi-repeat-search</b>	<b>N</b>		<b>WIDGETSTYLE</b>	Implementation style of completion widget		
<b>vi-replace</b>	<b>R</b>		Special characters in <b>bindkey</b> strings:			
<b>vi-replace-chars</b>	<b>r</b>		<b>\a</b>	Bell (alarm)		
<b>vi-rev-repeat-find</b>	<b>,</b>		<b>\b</b>	Backspace		
<b>vi-rev-repeat-search</b>	<b>“</b>		<b>\e, \E</b>	Escape		
<b>vi-set-buffer</b>	<b>m</b>		<b>\f</b>	Form feed		
<b>vi-set-mark</b>	<b>s</b>		<b>\n</b>	Newline		
<b>vi-substitute</b>	<b>~</b>		<b>\r</b>	Carriage return		
<b>vi-swap-case</b>	<b>u</b>		<b>\t</b>	Tab (horizontal)		
<b>vi-undo-change</b>	<b>&lt;</b>		<b>\v</b>	Tab (vertical)		
<b>vi-unindent</b>	<b>-</b>		<b>\nnn</b>	Octal character e.g. <b>\081</b>		
<b>vi-up-line-or-history</b>	<b>-</b>		<b>\xnn</b>	Hexadecimal character eg. <b>\x41</b>		
<b>vi-yank</b>	<b>y</b>					
<b>vi-yank-eol</b>	<b>Y</b>					
<b>vi-yank-whole-line</b>						