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.sunsite.dk/: 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 ash 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)

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 *listi*1 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:

`*`	Any string
`?`	Any character
`[`*class*`]`	Any single character from *class*
`[^`*class*`]`	Any single character not from class
`<`*num1*`-`*num2*`>`	Any number between *num1* and *num2*
	`<-`*num2*`>` from 0; `<`*num1*`->` to infinity.
`**/`	Directories to any level
`(`*pat1*`)`	Group patterns
`(`*pat1*`|`*pat2*`)`	*pat1* or *pat2* (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:

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

Extended patterns (option **EXTENDED_GLOB** must be set):

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

KSH_GLOB operators (patterns may contain `|` for alternatives):

`@(`*pat*`)`	Group patterns
`*(`*pat*`)`	Zero or more occurrences of *pat*
`+(`*pat*`)`	One or more occurrences of *pat*
`?(`*pat*`)`	Zero or one occurrences of *pat*
`!(`*pat*`)`	Anything but the pattern *pat*

Globbing flags with **EXTENDED_GLOB**:

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

Glob qualifiers (in parentheses after file name pattern):

`/`	Directory
`F`	Non-empty directory; for empty use `(/^F)`
`.`	Plain file
`@`	Symbolic link
`=`	Socket
`p`	Name pipe (FIFO)
`*`	Executable plain file
`%`	Special file
`%b`	Block special file
`%c`	Character special file
`r`	Readable by owner (N.B. not current user)
`w`	Writeable by owner
`x`	Executable by owner
`A`	Readable by members of file's group
`I`	Writeable by members of file's group
`E`	Executable by members of file's group
`R`	World readable
`W`	World writeable
`X`	World executable
`s`	Setuid
`S`	Setgid
`t`	Sticky bit
`f`*spec*	Has **chmod**-style permissions *spec*
`e`*string*	Evaluation *string* returns true status
`+`*cmd*	Same but *cmd* must be alphanumeric or _
`d`*dev*	Device number *dev* (major*256 + minor)
`l[-+]`*num*	Link count is (less than, greater than) *num*
`U`	Owned by current effective UID
`G`	Owned by current effective GID
`u`*uid*	Owned by given *uid* (may be `<`*name*`>`)
`g`*gid*	Owned by given *gid* (may be `<`*name*`>`)
`a[Mwhms][-+]n`	Access time in given units
`m[Mwhms][-+]n`	Modification time in given units
`c[Mwhms][-+]n`	Inode change time in given units
`^`	Negate following qualifiers
`-`	Toggle following links (first one turns on)
`M`	Mark directories
`T`	Mark directories, links, special files
`N`	Whole pattern expands to empty if no match
`D`	Leading dots may be matched
`n`	Sort numbers numerically
`o[nLlamcd]`	Order by given code (may repeat)
`O[nLlamcd]`	Order by reverse of given code
`[`*num*`]`	Select *num*th file in current order
`[`*num1*`,`*num2*`]`	Select *num1*th to *num2*th file (as arrays)
`:`*X*	History modifier *X*; may have more

Time units are Month, week, hour, minute, second.
Order codes are name (default), size, link count, access time, modification time, inode change time, directory depth.

Options

Set options with **setopt**, unset with **unsetopt**. Asterisk indicates on by default for native zsh.

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

OVERSTRIKE	Start line editor in overstrike mode
PATH_DIRS	*dir/cmd* can be found in **$PATH**
POSIX_BUILTINS	Illogical command behaviour
PRINT_EIGHT_BIT	Print all 8-bit characters directly
PRINT_EXIT_VALUE	Return status printed unless zero
PRIVILEGED	Special behaviour on setuid/setgid
PROMPT_BANG	Special treatment of **!** in prompt
***PROMPT_CR**	Prompt always at start of line
***PROMPT_PERCENT**	**%** escapes expanded in prompts
PROMPT_SUBST	**$** expansion etc. in prompts
PUSHD_IGNORE_DUPS	Don't push dir multiply on stack
PUSHD_MINUS	Reverse sense of **−** and **+** in **pushd**
PUSHD_SILENT	No non-err messages from **pushd**
PUSHD_TO_HOME	**pushd** with no argument goes to **~**
RC_EXPAND_PARAM	**X$***array* gives **X***elt1* **X***elt2* etc.
RC_QUOTES	**''** inside single quotes gives **'**
***RCS**	Run startup files
REC_EXACT	Exact completion matches are good
RESTRICTED	Shell has restricted capabilities
RM_STAR_SILENT	Don't warn on **rm ***
RM_STAR_WAIT	Wait before asking if **rm *** is OK
SHARE_HISTORY	Save and restore history per line
SH_FILE_EXPANSION	**~** etc. expansion done early
SH_GLOB	Disables non-extended zsh globs
SHIN_STDIN	Shell input comes from stdin
SH_NULL_CMD	Commandless redirections like **sh**
SH_OPTION_LETTERS	Single letter options are like **sh**
***SHORT_LOOPS**	**for** *words*; *list* works
SH_WORD_SPLIT	Split non-array variables yuckily
SINGLE_COMMAND	Execute one command then exit
SINGLE_LINE_ZLE	Line editing on single line (bad tty)
SUN_KEYBOARD_HACK	Unmatched ` at end of line ignored
TRANSIENT_RPROMPT	Right prompt goes away after edit
TRAPS_ASYNC	Traps may run when **wait**ing
TYPESET_SILENT	Silent on **typeset** *foo*
***UNSET**	Unset variables OK, treat as empty
VERBOSE	Output commands to be executed
VI	Same as **bindkey -v**
XTRACE	Show trace of execution with **$PS4**
ZLE	Line editor used to input lines

Option aliases (native zsh on right):

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

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

-0	**CORRECT**
-1	**PRINT_EXIT_VALUE**
-2	**NO_BAD_PATTERN**
-3	**NO_NO_MATCH**
-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**

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:

`!`*expr*	History expansion
alias	Alias expansion
`<(`*cmds*`)`	Replaced by file with output from *cmds*
`=(`*cmds*`)`	Same but can be reread (use for **diff**)
`>(`*cmds*`)`	Replaced by file with input to *cmds*
`$`*var*	Variable substitution
`${`*var*`}`	Same but protected, allows more options
`$(`*cmds*`)`	Replaced by output of *cmds*
`` ``*cmds*`` `	Older form of same, harder to nest
`$((`*expr*`))`	Arithmetic result of evaluating *expr*
X`{a,b}`*Y*	*X*`a`*Y* *X*`b`*y* (N.B. does no pattern matching)
X`{1..3}`*Y*	*X*`1`*Y* *X*`2`*y* *X*`3`*y*
X`{08..10}`*Y*	*X*`08`*Y* *X*`09`*y* *X*`10`*y*
`~`*user*, `~`*dir*	User home, named dir (*dir* is var name)
`=`*cmd*	/full/path/to/cmd
pattern	Glob file names, as above

History expansion:

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

Word selectors:

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

Modifiers on arguments (can omit word selector):

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

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

`${var:f`*m*`}`	Repeat modifier *m* till stops changing
`${var:F:`*N*`:`*m*`}`	Same but no more than *N* times
`${var:w`*m*`}`	Apply modifer *m* to words of string
`${var:W:`*sep*`:`*m*`}`	Same but words are separated by *sep*

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

%! %h	Current history event number
%#	**#** if superuser, else **%**
%%	A single **%**
%)	A **)** (use with **%***X***(.***tstr***.***fstr***)**)
%*	Time in 24-hour format with seconds
%/ %d	**$PWD**; *n* gives trailing parts, **-***n* leading
%c %. %C	Deprecated alternatives, differ by default *n*
%?	Return status of last command
%@ %t	Time of day in am/pm format
%B (**%b**)	Start (stop) bold face mode
%D %D{*str***}**	Date as *YY-MM-DD*, optional strftime spec
%E	Clear to end of line
%i	Script/function line number (**$LINENO**)
%j	Number of jobs as listed by jobs
%L	Shell depth (**$SHLVL**)
%l	Login terminal without **/dev** or **/dev/tty**
%M	Full host name
%m	Host name to first dot or *n* dots
%N	Name of script, function, sourced file
%n	Name of user (same as **$USERNAME**)
%S %s	Start (stop) standout mode
%T	Time of day, 24-hour format
%U %u	Start (stop) underline mode (patchy support)
%v	*n*th component of **$psvar** array
%W	Date as middle-endian *MM/DD/YY*
%w	Date as *DAY DD*
%y	Login terminal without **/dev**
%_	Parser state (continuation lines, debug)
%~	Like **%/**, **%d** but with tilde substitution
%{*esc***%}**	Escape sequence *esc* doesn't move cursor
%*X***(.***tstr***.***fstr***)**	*tstr* if test *X* gives *n*, else *fstr*
%<*str***<**	Truncate to n on left, *str* on left if so
%>*str***>**	Truncate to n on right, *str* 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:

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

Parameter flags in parentheses, immediately after left brace:

%	Expand **%**s in result as in prompts
@	Array expand even in double quotes
A	Create array parameter with **${...=...}**
a	Array index order, so **Oa** is reversed
c	Count characters for **${#***var***}**
C	Capitalize result
e	Do parameter, comand, arith expansion
f	Split result to array on newlines
F	Join arrays with newlines between elements
i	**oi** or **Oi** sort case independently
k	For associative array, result is keys
L	Lower case result
n	**on** or **On** sort numerically
o	Sort into ascending order
O	Sort into descending order
P	Interpret result as parameter name, get value
q	Quote result with backslashes
qq	Quote result with single quotes
qqq	Quote result with double quotes
qqqq	Quote result with **$'...'**
Q	Strip quotes from result
t	Output type of variable (see below)
u	Unique: remove duplicates after first
U	Upper case result
v	Include value in result; may have **(kv)**
V	Visible representation of special chars
w	Count words with **${#***var***}**
W	Same but empty words count
X	Report parsing errors (normally ignored)
z	Split to words using shell grammar
p	Following forms recognize print \-escapes
j:str:	Join words with *str* between
l:x:	Pad with spaces on left to width *x*
l:x::s1:	Same but pad with repeated *s1*
l:x::s1::s2:	Same but *s2* used once before any *s1*s
r:x::s1::s2:	Pad on right, otherwise same as **l** forms
s:str:	Split to array on occurrences of *str*
S	With patterns, search substrings
I:exp:	With patterns, match *exp*th occurrence
B	With patterns, include match beginning
E	With patterns, include match end
M	With patterns, include matched portion
N	With patterns, include length of match
R	With patterns, include unmatched part (rest)

Delimeters shown as **:***str***:** may be any pair of chars or matched parenthses **(***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**, **association**, then hyphen-separated words from following list:

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

ARGV0	Export to set name of external command
BAUD	Baud rate: compensation for slow terminals
†cdpath :	Directories searched for **cd** target
†COLUMNS	Width of screen
DIRSTACKSIZE	Maximum size of stack for **pushd**
ENV	File to source when started as **sh** or **ksh**
FCEDIT	Default editor used by **fc**
†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 **cd** target
†IFS	Characters that separate fields in words
KEYTIMEOUT	Time to wait for rest of key seq (1/100 s)
†LANG	Locale (usual variable, **LC_*** override)
†LC_ALL	Locale (overrides **LANG**, **LC_***)
†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 **$watch**
MAIL	Mail file to check (**$mailpath** overrides)
MAILCHECK	Mail check interval, secs (before prompt)
†mailpath :	List of files to check for new mail
†manpath :	Directories to find manual, used by **man**
†module_path :	Directories for **zmodload** 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 **PS**s
†PS2, PROMPT2	Printed for continuation lines
†PS3, PROMPT3	Print within **select** loop
†PS4, PROMPT4	For tracing execution (**xtrace** option)
†psvar :	Used with **%*n*v** 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 **read**
reply	Used to return array value
†RPS1, RPROMPT	Printed on right of screen for first line
†RPS2, RPROMPT2	Printed on right of screeen for continuation line
SAVEHIST	Max number of history lines saved
†SPROMPT	Prompt when correcting spelling
STTY	Export with **stty** arguments to command
†TERM	Type of terminal in use (**xterm** etc.)
TIMEFMT	Format for reporting usage with **time**
TMOUT	Send **SIGALRM** after seconds of inactivity
TMPPREFIX	Path prefix for shell's temporary files
†watch :	List of users or **all**, **notme** to watch for
WATCHFMT	Format of reports for **$watch**
†WORDCHARS	Chars considered parts of word by zle
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* **]]**:

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

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

-n	True if *str* has non-zero length
-o	True if option *str* is set
-t	True if *str* (number) is open file descriptor
-z	True if *str* has zero length

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

-nt	True if file *a* is newer than file *b*
-ot	True if file *a* is older than file *b*
-ef	True if *a* and *b* refer to same file (i.e. are linked)
=	True if string *a* matches pattern *b*
==	Same but more modern (and still not often used)
!=	True if string *a* does not match pattern *b*
<	True if string *a* sorts before string *b*
>	True if string *a* sorts after string *b*
-eq	True if numerical expressions *a* and *b* are equal
-ne	True if numerical expressions *a* and *b* are not equal
-lt	True if $a < b$ numerically
-gt	True if $a > b$ numerically
-le	True if $a \le b$ numerically
-ge	True if $a \ge b$ numerically

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

(*expr* **)**	Group tests
! *expr*	True if *expr* is false and vice versa
exprA **&&** *exprB*	True if both expressions true
exprA **||** *exprB*	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**, **cbrt**, **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 ))
```

Completion

Load new completion system with:

```
autoload -Uz compinit
compinit
```

Configuration: uses styles

zstyle *context style value…*

where context may be a pattern matching the following form:

:completion:*func*:*completer*:*cmd*:*arg*:*tag*

in which:

completion

Literal string always used by completion functions

func

Name of directly called widget, blank for contextual completion

completer

Method of completion e.g. **complete**; see below

cmd

Name of command being completed, or special command context

arg

Only valid with standard parsing: **arg-***n* for *n*th argument
option-opt-*n* for *n*th argument of option opt

tag

Indication of type of thing to be completed at this point.

Completers († indicates modifiers existing or later completions):

†_all_matches	Later completers add all matches
_approximate	Complete with errors in part so far
_complete	Basic completion
_correct	Correct word already typed
_expand	Perform shell expansions
_expand_alias	Expand aliases only
_history	Complete words from shell history
†_ignored	Reinstate matches omitted
†_list	List on first completion, insert on second
_match	Complete using patterns from line
†_menu	Menu completion, no menu selection
†_oldlist	Use existing list before generating new one
_prefix	Complete ignoring what's after cursor

Command contexts: any command name plus the special contexts:

-array-value-	Element in array
-brace-parameter-	Parameter within **${…}**
-assign-parameter-	Left hand side of assignment
-command-	Word in command position
-condition-	Word in **[[...]]** condition
-default-	Word with no specific completion
-equal-	Word beginning with equals sign
-first-	Tried first, may set **_compskip**
-math-	Inside arithmetic such as **((...))**
-parameter-	Parameter with bare **$** in front
-redirect-	Word after redirection operator
-subscript-	Inside parameter subscript
-tilde-	Between **~** and first **/** of argument
-value-	Right hand side of assignment

Tags:

accounts	For users-hosts style
all-expansions	When expanding, everything at once
all-files	All files rather than a subset
arguments	Command arguments
arrays	Names of array parameters
association-keys	Keys of associative arrays
bookmarks	Bookmarks for URLs, ZFTP, etc.
builtins	Names of builtin commands
characters	Character classes, stty characters
colormapids	X colormap IDs
colors	Names of colors, usually X
commands	External commands, subcommands
contexts	Contexts in zstyle
corrections	Possible approximations, corrections
cursors	X cursor names
default	Nothing specific in certain contexts
descriptions	Used in format style for matches
devices	Device special files
directories	Directories
directory-stack	Entries in pushd directory stack
displays	X displays
domains	Network domain (DNS) names
expansions	Individual expansions instead of all
file-descriptors	Numbers of open file descriptors
files	Generic file matching tag
fonts	X font names
fstypes	Files system types for mount etc.
functions	Shell functions, possibly other types
globbed-files	Names of files matched by pattern
groups	UNIX groups
history-words	Words from shell history
hosts	Names of network hosts
indexes	Indexes of arrays
jobs	Shell jobs
interfaces	Network interfaces (as from ifconfig)
keymaps	ZLE keymaps
keysyms	Names of X keysyms
libraries	Names of system libraries
limits	System resource limits
local-directories	Subdirectories of current directories
manuals	Names of manual pages
mailboxes	E-mail folders
maps	NIS maps etc.
messages	Used in format style for messages
modifiers	X modifiers
modules	Shell modules etc.
my-accounts	Own accounts, with users-hosts style
named-directories	Directories named by a parameter
names	Names of all sorts
newsgroups	USENET newgroups
nicknames	Nicknames of NIS maps
options	Options to commands
original	Original when correcting, expanding
other-accounts	Other accounts with users-hosts style

Tags continued:

packages	RPM, Debian etc. packages
parameters	Names of shell parameters
path-directories	Directories under $cdpath
paths	Used with assorted directory paths
pods	Perl documentation
ports	TCP, UDP prots
prefixes	URL etc. prefixes
printers	Names of print queues
processes	PIDs
processes-names	Names of processes in killall
sequences	MH sequences etc.
sessions	ZFTP sessions etc.
signals	System signal names, HUP etc.
strings	Assorted strings, e.g. second arg of cd
styles	Styles in zstyle
suffixes	Filename extensions
tags	Tags used with rpm etc.
targets	Targets inside Makefiles
time-zones	Time zones with TZ parameter etc.
types	Assorted types of anything
urls	Used with web addresses
users	Names of users
values	Values in lists
variant	Used when picking variant of command
visuals	X visuals
warnings	Used in the format style for warnings
widgets	Names of zsh widgets
windows	IDs of X windows
zsh-options	Shell options

Styles († indicates on by default):

accept-exact	Accept exact match even if ambiguous
†add-space	Add a space after expansions
ambiguous	Cursor after ambiguous path component
assign-list	**PATH**-style list on assignment
auto-description	String for option descs without specific
avoid-completer	Avoid completer with **_all_matches**
cache-path	Path to top of various caches
cache-policy	Function to decide on cache rebuilding
call-command	If true, use external (slow) command
command	External command to call (+args)
command-path	Override **PATH** for commands to match
commands	Default sys init commands (start etc.)
complete	Complete aliases (**_expand_alias**)
completer	The list of completers to try (see above)
†condition	Delay insertion of matches (_list)
disabled	Disabled aliases (**_expand_alias**)
disable-stat	If set, _cvs uses ls instead of zsh/stat
domains	Net domains (**/etc/resolv.conf**)
expand	For **prefix**, **suffix** in multiple parts
fake	Add *value*:*desc* fake completions
fake-files	*dir*:*names* add *names* in *dir*
fake-parameters	Params to complete even if not yet set
file-patterns	*pattern*:*tag* generates files with tag
file-sort	size, links, time, access, inode, reverse
filter	In LDAP, attributes for filtering
force-list	Just list matches: **always** or number
format	Desc string, **%d** shows specific desc
†glob	Attempt glob expansion (**_expand**)
†global	Global aliases (**_expand_alias**)
group-name	Name groups shown together by tag
group-order	Order groups shown together by tag
groups	Unix groups, as per **/etc/group**
hidden	Complete but don't list matches
hosts	List of host names, as **/etc/hosts**
hosts-ports	List of *hosts*:*ports* for TCP/UDP
ignore-line	Don't complete words already present
ignore-parents	**parent** or **pwd**: ignore parent dirs
ignored-patterns	If pattern matched, don't complete
insert	All matches at once (**_all_matches**)
insert-ids	Convert **%***cmd* to unambiguous PID
insert-tab	Insert TAB if no non-whitespace yet
insert-unambiguous	Only menu complete when no prefix to insert
keep-prefix	Try to keep expandable prefix
last-prompt	Return to last editing line if possible
list	Control listing when history completing
list-colors	Color specs like **LS_COLORS**
†list-grouped	Grouped listing shown more compactly
list-packed	All matches shown more compactly
list-prompt	Prompt when scrolling completions
list-rows-first	Increment rows first in lists
list-suffixes	Show ambiguous bits of multiple paths
list-separator	Separates description in verbose list
local	*host*:*path*:*dir* for URLs as files
mail-directory	Directory for mailbox files (**~/Mail**)
match-original	Add * when matching (**_match**)
matcher	Apply match control syntax per tag
matcher-list	Apply match control syntax globally
max-errors	Max errors allowed in approx/correct
max-matches-width	Cols to reserve for matches (not desc)
menu	Use menu completion
muttrc	Alternative for **~/.muttrc**
numbers	Prefer job numbers instead of name
old-list	Retain list of matches (**_oldlist**)
old-matches	Use old match list (**_all_matches**)
old-menu	Keep list for meus (**_oldlist**)
original	Add original match for approx/correct
packageset	For arguments of Debian **dpkg**
path	For X colors, path to **rgb.txt**
pine-directory	Directory for PINE mailboxes
ports	TCP/IP services (**/etc/services**)
prefix-hidden	Hide common prefix e.g. in options
prefix-needed	Common prefix must by typed by user
preserve-prefix	Initial file patterns to leave alone
range	Range of words in history to consider
†regular	Complete regular aliases

Styles continued:

†remote-access	Control remote access for e.g. **_cvs**
remove-all-dups	Never complete duplicates in history
select-prompt	Prompt shown in menu selection
select-scroll	Lines to scroll in menu selection
separate-sections	Manual sections used as part of tag
show-completer	Show progress of completers as msg
single-ignored	Control **_ignore** when single match
sort	Override sorting of matches
special-dirs	Add **.** and **..** to file list
squeeze-slashes	**fo//ba** is **fo/ba** not **fo/*/ba**
stop	Pause before looping shell history
strip-comments	Remove display name from email addr
subst-globs-only	Only take expansions from globbing
†substitute	When expanding, first try subst
†suffix	Only expand path with no **/suffix**
tag-order	Preference order for tags in context
urls	Determine where URLs are taken from
use-cache	Control caching for various commands
use-compctl	Use **comptl**-style completions
use-perl	Use simpler Perl code for **_make**
users	List of user names
users-hosts	List of *user@host* possibilities
users-hosts-ports	List of *user@host*:*port*
†verbose	Verbose output e.g. option descriptions
word	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 *spec*s, 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		
accept-and-infer-next-history			
accept-and-menu-complete			
accept-line	^M	^M	^M
accept-line-and-down-history	^O		
argument-base			
backward-char	^B		
backward-delete-char	^H		
backward-delete-word			
backward-kill-line			
backward-kill-word	^W		
backward-word	€b		
beep			
beginning-of-buffer-or-history	€<		
beginning-of-history			
beginning-of-line	^A		
beginning-of-line-hist			
capitalize-word	€c		
clear-screen	^L	^L	^L
complete-word			
copy-prev-word	€^_		
copy-prev-shell-word			
copy-region-as-kill	€w		
delete-char			
delete-char-or-list	^D		
delete-word			
describe-key-briefly			
digit-argument	€0 ..1..9		
down-case-word	€l		
down-history		^n	
down-line-or-history	^n	j	*down*
down-line-or-search			
emacs-backward-word			
emacs-forward-word			
end-of-buffer-or-history		€>	
end-of-history			
end-of-line	^E		
end-of-line-hist			
end-of-list			
exchange-point-and-mark	^X^X		
execute-last-named-cmd	€x		
execute-name-cmd	€x		
expand-cmd-path			
expand-history	€!		
expand-or-complete	^I		^I
expand-or-complete-prefix			
expand-word	^X*		
forward-char		^F	
forward-word	€f		
get-line	€g		
gosmacs-transpose-chars			
history-beginning-search-backward			
history-beginning-search-forward			
history-incremental-search-backward	^R ^Xr		
history-incremental-search-forward	^S ^Xs		
history-search-backward	€p		
history-search-forward	€n		
infer-next-history	^x^n		
insert-last-word	€_		
kill-buffer	^X^K		
kill-line	^K		
kill-region			
kill-whole-line	^U		
kill-word	€d		
list-choices	€d	^d	^d
list-expand	^Xg	^G	^G
magic-space			
menu-complete			
menu-expand-or-complete			
neg-argument	€		
overwrite-mode	^X^O		
pound-insert		#	
push-input			
push-line	€q		
push-line-or-edit			
quoted-insert	^V		
quote-line	€'		
quote-region	€"		
recursive-edit			
redisplay		^R	^R
redo			
reset-prompt			
reverse-menu-complete			
run-help	€h		
self-insert	...		...
self-insert-unmeta	€^M		
send-break	^G		
set-mark-command	^@		
spell-word	€$		
set-local-history			
transpose-chars	^T		
transpose-words	€t		
undefined-key			
undo	^_		
universal-argument			
up-case-word	€u		
up-history		^p	
up-line-or-history	^p	k	*up*
up-line-or-search			
vi-add-eol		A	
vi-add-next		a	
vi-backward-blank-word		B	
vi-backward-char		h	^H *left*
vi-backward-delete-char		X	^H
vi-backward-kill-word			^W
vi-backward-word		b	
vi-beginning-of-line			
vi-caps-lock-panic			
vi-change		c	
vi-change-eol		C	
vi-change-whole-line		S	
vi-cmd-mode	^XV		€
vi-delete		d	
vi-delete-char		x	
vi-digit-or-beginning-of-line		0	
vi-down-line-or-history		+	

Builtin widgets cont.:

vi-end-of-line		**$**
vi-fetch-history		**G**
vi-find-next-char	**^X^F**	**f**
vi-find-next-char-skip		**t**
vi-find-prev-char		**F**
vi-find-prev-char-skip		**T**
vi-first-non-blank		**^**
vi-forward-blank-word		**W**
vi-forward-blank-word-end		**E**
vi-forward-char		**l** *rght*
vi-forward-word		**w**
vi-forward-word-end		**e**
vi-goto-column	**€|**	**|**
vi-goto-mark		**`**
vi-goto-mark-line		**.**
vi-history-search-backward		**/**
vi-history-search-forward		**?**
vi-indent		**>**
vi-insert		**i**
vi-insert-bol		**I**
vi-join	**^X^J**	**J**
vi-kill-eol		**D**
vi-kill-line		**^U**
vi-match-bracket	**^X^B**	**%**
vi-open-line-above		**O**
vi-open-line-below		**o**
vi-oper-swap-case		
vi-pound-insert		
vi-put-after		**P**
vi-put-before		**p**
vi-quoted-insert		**^V**
vi-repeat-change		**.**
vi-repeat-find		**;**
vi-repeat-search		**N**
vi-replace		**R**
vi-replace-chars		**r**
vi-rev-repeat-find		**,**
vi-rev-repeat-search		
vi-set-buffer		**“**
vi-set-mark		**m**
vi-substitute		**s**
vi-swap-case		**~**
vi-undo-change		**u**
vi-unindent		**<**
vi-up-line-or-history		**-**
vi-yank		**y**
vi-yank-eol		**Y**
vi-yank-whole-line		
what-cursor-position	**^X=**	
where-is		
which-command	**€**	
yank	**^y**	
yank-pop	**€**	

Special parameters inside user-defined widgets; † indicates readonly:

BUFFER	Entire editing buffer
BUFFERLINES	Number of screen lines for full buffer
†CONTEXT	**start**, **cont**, **select**, **vared**
CURSOR	Index of cursor position into **$BUFFER**
CUTBUFFER	Last item to be killed
HISTNO	Currently history line being retrieved
†KEYMAP	Currently selected keymap
†KEYS	Keys typed to invoke current widget
killring	Array of previously killed items, can resize
†LASTSEARCH	Last search string in interactive search
†LASTWIDGET	Last widget to be executed
LBUFFER	Part of buffer left of cursor
MARK	Index of mark position into **$BUFFER**
NUMERIC	Numeric argument passed with widget
†PENDING	Number of bytes still to be read
†PREBUFFER	Input already read (no longer being edited)
PREDISPLAY	Text to display before editable buffer
POSTDISPLAY	Text to display after editable buffer
RBUFFER	Part of buffer starting from cursor
WIDGET	Name of widget being executed
WIDGETFUNC	Name of function implementing **$WIDGET**
WIDGETSTYLE	Implementation style of completion widget

Special characters in **bindkey** strings:

\a	Bell (alarm)
\b	Backspace
\e, **\E**	Escape
\f	Form feed
\n	Newline
\r	Carriage return
\t	Tab (horizontal)
\v	Tab (vertical)
*****nnn*	Octal character e.g **\081**
\x*nn*	Hexadecimal character eg. **\x41**
\M*x*, **\M-***x*	Set 8th bit in character
\C*x*, **\C-***x*	Control character e.g. **\C-a**
^*x*	Control character e.g. **^a** (same as **^A**)
^?	Delete
****	Single backslash

Keymaps:

emacs	Like Emacs editor
viins	Like Vi editor in insert mode
vicmd	Like Vi editor in command mode
.safe	Emergency keymap, not modifiable