

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)

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:

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

Mailing lists

zsh-users@sunsite.dk: users' mailing list for general questions and tips; to join, mail zsh-users-subscribe@sunsite.dk.

zsh-workers@sunsite.dk: mailing list for bug reports, patches and developers' discussions; to join, mail zsh-workers-subscribe@sunsite.dk. 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:

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

a-b	A character in the range <i>a</i> to <i>b</i>
[: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 <i>pat</i>
pat1^pat2	Match <i>pat1</i> then anything other than <i>pat2</i>
pat1~pat2	Anything matching <i>pat1</i> but not <i>pat2</i>
X#	Zero or more occurrences of element <i>X</i>
X##	One or more occurrences of element <i>X</i>

KSH_GLOB operators (patterns may contain | for alternatives):

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

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.
(#anum)	Match with <i>num</i> approximations
(#s)	Match only at start of test string
(#e)	Match only at end of test string
(#qexpr)	<i>expr</i> is 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	Writable by owner
x	Executable by owner
A	Readable by members of file's group
I	Writable 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
fspec	Has chmod -style permissions <i>spec</i>
estring	Evaluation <i>string</i> returns true status
+cmd	Same but <i>cmd</i> must be alphanumeric or <code>_</code>
ddev	Device number <i>dev</i> (major*256 + minor)
l[-+]num	Link count is (less than, greater than) <i>num</i>
U	Owned by current effective UID
G	Owned by current effective GID
uuid	Owned by given <i>uid</i> (may be <i><name></i>)
ggid	Owned by given <i>gid</i> (may be <i><name></i>)
a[Mwhms] [-+]n	Access time in given units (see below)
m[Mwhms] [-+]n	Modification time in given units
c[Mwhms] [-+]n	Inode change time in given units
L[kmp] [-+]n	Size in given units (see below)
^	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 (as below; may repeat)
O[nLlamcd]	Order by reverse of given code
[num]	Select <i>num</i> th file in current order
[num1, num2]	Select <i>num1</i> th to <i>num2</i> th file (as arrays)
:X	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 setopt , unset with unsetopt . Asterisk indicates on by default for native zsh.			
*ALIASES	Expand aliases	CSH_NULL_GLOB	One glob must succeed, failures go
ALL_EXPORT	Export all variables to environment	DVORAK	Dvorak keyboard for correction
*ALWAYS_LAST_PROMPT	Completion lists after prompt	EMACS	Same as bindkey -e
ALWAYS_TO_END	On completion go to end of word	*EQUALS	Expand =cmd to /path/to/cmd
*APPEND_HISTORY	History appends to existing file	ERR_EXIT	Exit shell on non-zero status
AUTO_CD	Directory as command does cd	ERR_RETURN	Return from function instead
AUTO_CONTINUE	Jobs are continued when disowned	*EVAL_LINE_NO	\$LINENO counts inside eval code
*AUTO_LIST	List ambiguous completions	*EXEC	Execute commands
*AUTO_MENU	Menu complete after two tabs	EXTENDED_GLOB	See globbing section above
AUTO_NAME_DIRS	Variables always can be %~ abbrevs	EXTENDED_HISTORY	Timestamps saved to history file
*AUTO_PARAM_KEYS	Magic completion for parameters	*FLOW_CONTROL	Use ^S/^Q style flow control
*AUTO_PARAM_SLASH	\$dirname completes with /	*FUNCTION_ARGZERO	\$0 in function is its name
AUTO_PUSHD	cd uses directory stack too	*GLOB	Use globbing as described above
*AUTO_REMOVE_SLASH	Trailing / in completion removed	*GLOBAL_EXPORT	Exported variables not made local
AUTO_RESUME	cmd can resume job %cmd	*GLOBAL_RCS	Execute /etc/z* files
BAD_PATTERN	Errors on pattern syntax; else literal	GLOB_ASSIGN	var= expands, assigns array
*BANG_HIST	! style history allowed	GLOB_COMPLETE	Patterns are active in completion
*BARE_GLOB_QUAL	Glob qualifiers with bare parens	GLOB_DOTS	Patterns may match leading dots
BASH_AUTO_LIST	List completions on second tab	GLOB_SUBST	Substituted characters may glob
*BEEP	Beep on all errors	*HASH_CMDS	Store command location for speed
*BG_NICE	Background jobs at lower priority	*HASH_DIRS	Store for all commands in dir
BRACE_CCL	X{ab} expands to Xa Xb	*HASH_LIST_ALL	Store all on first completion
BSD_ECHO	No echo escapes unless -e given	HIST_ALLOW_CLOBBER	On clobber error, up arrow to retry
*CASE_GLOB	Glob case sensitively	*HIST_BEEP	Beep when going beyond history
C_BASES	Output hexadecimal with 0x	HIST_EXPIRE_DUPS_FIRST	Duplicate history entries lost first
CDABLE_VARS	cd var works if \$var is directory	HIST_FIND_NO_DUPS	History search finds once only
CHASE_DOTS	Resolve .. in cd	HIST_IGNORE_ALL_DUPS	Remove all earlier duplicate lines
CHASE_LINKS	Resolve symbolic links in cd	HIST_IGNORE_DUPS	Remove duplicate of previous line
*CHECK_JOBS	Check jobs before exiting shell	HIST_IGNORE_SPACE	Don't store lines starting with space
*CLOBBER	Allow redirections to overwrite	HIST_NO_FUNCTIONS	Don't store shell functions
COMPLETE_ALIASES	Completion uses unexpanded aliases	HIST_NO_STORE	Don't store history and fc
COMPLETE_IN_WORD	Completion works inside words	HIST_REDUCE_BLANKS	Trim multiple insignificant blanks
CORRECT	Correct spelling of commands	HIST_SAVE_NO_DUPS	Remove duplicates when saving
CORRECT_ALL	Correct spelling of all arguments	HIST_VERIFY	Show ! history line for editing
CSH_JUNKIE_HISTORY	Single ! for previous command	*HUP	Send SIGHUP to processes on exit
CSH_JUNKIE_LOOPS	list; end for do...done	IGNORE_BRACES	Don't use {a,b} expansions
CSH_JUNKIE_QUOTES	No newlines in quotes	IGNORE_EOF	Ignore ^D (stty eof char)
CSH_NULLCMD	Redirections with no commands fail	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 Xelt1 Xelt2 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 waiting
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 <i>cmds</i>
=(cmds)	Same but can be reread (use for diff)
>(cmds)	Replaced by file with input to <i>cmds</i>
\$var	Variable substitution
\${var}	Same but protected, allows more options
\$(cmds)	Replaced by output of <i>cmds</i>
`cmds`	Older form of same, harder to nest
\$((expr))	Arithmetic result of evaluating <i>expr</i>
X{a,b}Y	<i>XaY XbY</i> (N.B. does no pattern matching)
X{1..3}Y	<i>X1Y X2Y X3Y</i>
X{08..10}Y	<i>X08Y X09Y X10Y</i>
~user, ~dir	User home, named dir (<i>dir</i> 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 <i>cmd</i>
!?str	Last command containing <i>str</i>
!#	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 <i>s1</i> by <i>s2</i>
!!:1:gs/s2/s2/	Same but global
!!:1:&	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:

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

%! %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 ; <i>n</i> gives trailing parts, <i>-n</i> leading
%c % . %C	Deprecated alternatives, differ by default <i>n</i>
??	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 <i>YY-MM-DD</i> , 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 <i>n</i> 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	<i>n</i> th component of \$psvar array
%W	Date as middle-endian <i>MM/DD/YY</i>
%w	Date as <i>DAY DD</i>
%y	Login terminal without /dev
%_	Parser state (continuation lines, debug)
%~	Like %/ , %d but with tilde substitution
{esc%}	Escape sequence <i>esc</i> doesn't move cursor
%X(.tstr.fstr)	<i>tstr</i> if test <i>X</i> gives <i>n</i> , else <i>fstr</i>
%<str<	Truncate to <i>n</i> on left, <i>str</i> on left if so
%>str>	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:

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

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 <code>\${...=...}</code>
a	Array index order, so 0a is reversed
c	Count characters for <code>\${#var}</code>
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 <code>'...'</code>
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 <code>\${#var}</code>
W	Same but empty words count
X	Report parsing errors (normally ignored)
z	Split to words using shell grammar
p	Following forms recognize print <code>\-</code> escapes
j:str:	Join words with <i>str</i> between
l:x:	Pad with spaces on left to width <i>x</i>
l:x::s1:	Same but pad with repeated <i>s1</i>
l:x::s1::s2:	Same but <i>s2</i> used once before any <i>s1</i> s
r:x::s1::s2:	Pad on right, otherwise same as l forms
s:str:	Split to array on occurrences of <i>str</i>
S	With patterns, search substrings
I:exp:	With patterns, match <i>exp</i> 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)

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:

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

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

-n	True if <i>str</i> has non-zero length
-o	True if option <i>str</i> is set
-t	True if <i>str</i> (number) is open file descriptor
-z	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'`:

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

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

(expr)	Group tests
! expr	True if <i>expr</i> 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**, **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 ))
```

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

sequences	MH sequences etc.	fake-files	<i>dir:names</i> add <i>names</i> in <i>dir</i>	muttrc	Alternative for <i>~/muttrc</i>
sessions	ZFTP sessions etc.	fake-parameters	Params to complete even if not yet set	numbers	Prefer job numbers instead of name
signals	System signal names, HUP etc.	file-patterns	<i>pattern:tag</i> generates files with tag	old-list	Retain list of matches (_oldlist)
strings	Assorted strings, e.g. second arg of <i>cd</i>	file-sort	size, links, time, access, inode, reverse	old-matches	Use old match list (_all_matches)
styles	Styles in <i>zstyle</i>	filter	In LDAP, attributes for filtering	old-menu	Keep list for <i>meus</i> (_oldlist)
suffixes	Filename extensions	force-list	Just list matches: always or number	original	Add original match for <i>approx/correct</i>
tags	Tags used with <i>rpm</i> etc.	format	Desc string, %d shows specific desc	packageset	For arguments of Debian dpkg
targets	Targets inside Makefiles	fglob	Attempt glob expansion (_expand)	path	For X colors, path to rgb.txt
time-zones	Time zones with TZ parameter etc.	fglobal	Global aliases (_expand_alias)	pine-directory	Directory for PINE mailboxes
types	Assorted types of anything	group-name	Name groups shown together by tag	ports	TCP/IP services (/etc/services)
urls	Used with web addresses	group-order	Order groups shown together by tag	prefix-hidden	Hide common prefix e.g. in options
users	Names of users	groups	Unix groups, as per /etc/group	prefix-needed	Common prefix must be typed by user
values	Values in lists	hidden	Complete but don't list matches	preserve-prefix	Initial file patterns to leave alone
variant	Used when picking variant of command	hosts	List of host names, as /etc/hosts	range	Range of words in history to consider
visuals	X visuals	hosts-ports	List of <i>hosts:ports</i> for TCP/UDP	regular	Complete regular aliases
warnings	Used in the format style for warnings	ignore-line	Don't complete words already present	Styles continued:	
widgets	Names of zsh widgets	ignore-parents	parent or pwd : ignore parent dirs	tremote-access	Control remote access for e.g. _cvs
windows	IDs of X windows	ignored-patterns	If pattern matched, don't complete	remove-all-dups	Never complete duplicates in history
zsh-options	Shell options	insert	All matches at once (_all_matches)	select-prompt	Prompt shown in menu selection
		insert-ids	Convert <i>%cmd</i> to unambiguous PID	select-scroll	Lines to scroll in menu selection
Styles († indicates on by default):		insert-tab	Insert TAB if no non-whitespace yet	separate-sections	Manual sections used as part of tag
accept-exact	Accept exact match even if ambiguous	insert-unambiguous	Only menu complete when no prefix to insert	show-completer	Show progress of completers as <i>msg</i>
†add-space	Add a space after expansions	keep-prefix	Try to keep expandable prefix	single-ignored	Control _ignore when single match
ambiguous	Cursor after ambiguous path component	last-prompt	Return to last editing line if possible	sort	Override sorting of matches
assign-list	PATH -style list on assignment	list	Control listing when history completing	special-dirs	Add . and .. to file list
auto-description	String for option desc's without specific	list-colors	Color specs like LS_COLORS	squeeze-slashes	fo//ba is fo/ba not fo/*/ba
avoid-completer	Avoid completer with _all_matches	†list-grouped	Grouped listing shown more compactly	stop	Pause before looping shell history
cache-path	Path to top of various caches	list-packed	All matches shown more compactly	strip-comments	Remove display name from email addr
cache-policy	Function to decide on cache rebuilding	list-prompt	Prompt when scrolling completions	subst-globs-only	Only take expansions from globbing
call-command	If true, use external (slow) command	list-rows-first	Increment rows first in lists	†substitute	When expanding, first try subst
command	External command to call (+args)	list-suffixes	Show ambiguous bits of multiple paths	†suffix	Only expand path with no /suffix
command-path	Override PATH for commands to match	list-separator	Separates description in verbose list	tag-order	Preference order for tags in context
commands	Default sys init commands (start etc.)	local	<i>host:path:dir</i> for URLs as files	urls	Determine where URLs are taken from
complete	Complete aliases (_expand_alias)	mail-directory	Directory for mailbox files (~/Mail)	use-cache	Control caching for various commands
completer	The list of completers to try (see above)	match-original	Add * when matching (_match)	use-compctl	Use compctl -style completions
†condition	Delay insertion of matches (_list)	matcher	Apply match control syntax per tag	use-perl	Use simpler Perl code for _make
disabled	Disabled aliases (_expand_alias)	matcher-list	Apply match control syntax globally	users	List of user names
disable-stat	If set, _cvs uses <i>ls</i> instead of <i>zsh/stat</i>	max-errors	Max errors allowed in <i>approx/correct</i>	users-hosts	List of <i>user@host</i> possibilities
domains	Net domains (/etc/resolv.conf)	max-matches-width	Cols to reserve for matches (not desc)	users-hosts-ports	List of <i>user@host:port</i>
expand	For prefix , suffix in multiple parts	menu	Use menu completion	†verbose	Verbose output e.g. option descriptions
fake	Add <i>value:desc</i> fake completions			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 **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;				<code>execute-last-named-cmd</code>	<code>€z</code>			<code>redisplay</code>	<code>^R</code> <code>^R</code>
€ denotes escape key:				<code>execute-name-cmd</code>	<code>€x</code>			<code>redo</code>	
<code>accept-and-hold</code>	<code>€a</code>			<code>expand-cmd-path</code>				<code>reset-prompt</code>	
<code>accept-and-infer-next-history</code>				<code>expand-history</code>	<code>€!</code>			<code>reverse-menu-complete</code>	
<code>accept-and-menu-complete</code>				<code>expand-or-complete</code>	<code>^I</code>	<code>^I</code>		<code>run-help</code>	<code>€h</code>
<code>accept-line</code>				<code>expand-or-complete-prefix</code>				<code>self-insert</code>	<code>...</code> <code>...</code>
<code>accept-line-and-down-history</code>	<code>^M</code>	<code>^M</code>	<code>^M</code>	<code>expand-word</code>	<code>^X*</code>			<code>self-insert-unmeta</code>	<code>€^M</code>
<code>argument-base</code>	<code>^O</code>			<code>forward-char</code>		<code>^F</code>		<code>send-break</code>	<code>^G</code>
<code>backward-char</code>	<code>^B</code>			<code>forward-word</code>	<code>€f</code>			<code>set-mark-command</code>	<code>^@</code>
<code>backward-delete-char</code>	<code>^H</code>			<code>get-line</code>	<code>€g</code>			<code>spell-word</code>	<code>€s</code>
<code>backward-delete-word</code>				<code>gosmacs-transpose-chars</code>				<code>set-local-history</code>	
<code>backward-kill-line</code>				<code>history-beginning-search-backward</code>				<code>transpose-chars</code>	<code>^T</code>
<code>backward-kill-word</code>	<code>^W</code>			<code>history-beginning-search-forward</code>				<code>transpose-words</code>	<code>€t</code>
<code>backward-word</code>	<code>€b</code>			<code>history-incremental-search-backward</code>	<code>^R</code>			<code>undefined-key</code>	<code>^_</code>
<code>beep</code>				<code>history-incremental-search-forward</code>	<code>^Xr</code>			<code>undo</code>	
<code>beginning-of-buffer-or-history</code>	<code>€<</code>			<code>history-incremental-search-backward</code>	<code>^S</code>			<code>universal-argument</code>	<code>€u</code>
<code>beginning-of-history</code>				<code>history-incremental-search-forward</code>	<code>^Xs</code>			<code>up-case-word</code>	
<code>beginning-of-line</code>	<code>^A</code>			<code>history-search-backward</code>	<code>€p</code>			<code>up-history</code>	<code>^p</code>
<code>beginning-of-line-hist</code>				<code>history-search-forward</code>	<code>€n</code>			<code>up-line-or-history</code>	<code>^p</code> <code>k</code> <code>up</code>
<code>capitalize-word</code>	<code>€c</code>			<code>infer-next-history</code>	<code>^x^n</code>			<code>up-line-or-search</code>	
<code>clear-screen</code>	<code>^L</code>	<code>^L</code>	<code>^L</code>	<code>insert-last-word</code>	<code>€_</code>			<code>vi-add-eol</code>	<code>A</code>
<code>complete-word</code>				<code>kill-buffer</code>	<code>€_</code>			<code>vi-add-next</code>	<code>a</code>
<code>copy-prev-word</code>	<code>€^_</code>			<code>kill-line</code>	<code>^X^K</code>			<code>vi-backward-blank-word</code>	<code>B</code>
<code>copy-prev-shell-word</code>				<code>kill-region</code>	<code>^K</code>			<code>vi-backward-char</code>	<code>h</code> <code>^H</code> <i>left</i>
<code>copy-region-as-kill</code>	<code>€w</code>			<code>kill-whole-line</code>				<code>vi-backward-delete-char</code>	<code>X</code> <code>^H</code> <code>^W</code>
<code>delete-char</code>				<code>kill-word</code>	<code>^U</code>			<code>vi-backward-kill-word</code>	<code>^W</code>
<code>delete-char-or-list</code>	<code>^D</code>			<code>list-choices</code>	<code>€d</code>	<code>^d</code> <code>^d</code>		<code>vi-backward-word</code>	<code>b</code>
<code>delete-word</code>				<code>list-expand</code>	<code>€d</code>	<code>^d</code> <code>^d</code>		<code>vi-beginning-of-line</code>	
<code>describe-key-briefly</code>				<code>magic-space</code>	<code>€Xg</code>	<code>^G</code> <code>^G</code>		<code>vi-caps-lock-panic</code>	
<code>digit-argument</code>	<code>€0..1..9</code>			<code>menu-complete</code>				<code>vi-change</code>	<code>c</code>
<code>down-case-word</code>	<code>€l</code>			<code>menu-expand-or-complete</code>				<code>vi-change-eol</code>	<code>C</code>
<code>down-history</code>		<code>^n</code>		<code>neg-argument</code>	<code>€-</code>			<code>vi-change-whole-line</code>	<code>S</code>
<code>down-line-or-history</code>	<code>^n</code>	<code>j</code>	<i>down</i>	<code>overwrite-mode</code>	<code>^X^O</code>			<code>vi-cmd-mode</code>	<code>^XV</code> <code>€</code>
<code>down-line-or-search</code>				<code>pound-insert</code>		<code>#</code>		<code>vi-delete</code>	<code>d</code>
<code>emacs-backward-word</code>				<code>push-input</code>				<code>vi-delete-char</code>	<code>x</code>
<code>emacs-forward-word</code>				<code>push-line</code>	<code>€q</code>			<code>vi-digit-or-beginning-of-line</code>	<code>0</code>
<code>end-of-buffer-or-history</code>	<code>€></code>			<code>push-line-or-edit</code>				<code>vi-down-line-or-history</code>	<code>+</code>
<code>end-of-history</code>				<code>quoted-insert</code>	<code>^V</code>			Builtin widgets cont.:	
<code>end-of-line</code>	<code>^E</code>			<code>quote-line</code>	<code>€'</code>			<code>vi-end-of-line</code>	<code>\$</code>
<code>end-of-line-hist</code>				<code>quote-region</code>	<code>€"</code>			<code>vi-fetch-history</code>	<code>G</code>
<code>end-of-list</code>				<code>recursive-edit</code>				<code>vi-find-next-char</code>	<code>^X^F</code> <code>f</code>
<code>exchange-point-and-mark</code>	<code>^X^X</code>							<code>vi-find-next-char-skip</code>	<code>t</code>
								<code>vi-find-prev-char</code>	<code>F</code>

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