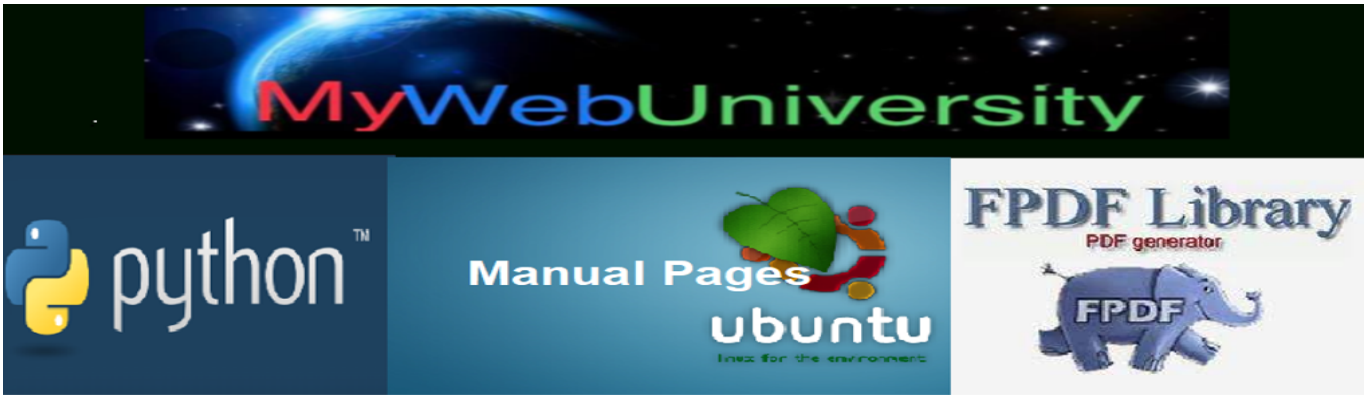

Rocky Enterprise Linux 9.2 Manual Pages on command 'zshcompctl.1'

$ man zshcompctl.1

ZSHCOMPCTL(1) General Commands Manual ZSHCOMPCTL(1)

NAME

zshcompctl - zsh programmable completion

DESCRIPTION

This version of zsh has two ways of performing completion of words on the command line. New users of the shell may prefer to use the newer and more powerful system based on shell functions; this is described in zshcompsys(1), and the basic shell mechanisms which sup? port it are described in zshcompwid(1). This manual entry describes the older compctl command.

compctl [-CDT] options [command ...]

compctl [-CDT] options [-x pattern options - ... --]

[+ options [-x ... --] ... [+]] [command ...]

compctl -M match-specs ...

compctl -L [-CDTM] [command ...]

compctl + command ...

Control the editor's completion behavior according to the supplied set of options. Vari? ous editing commands, notably expand-or-complete-word, usually bound to tab, will attempt to complete a word typed by the user, while others, notably delete-char-or-list, usually bound to ^D in EMACS editing mode, list the possibilities; compctl controls what those possibilities are. They may for example be filenames (the most common case, and hence the default), shell variables, or words from a user-specified list.

COMMAND FLAGS

Completion of the arguments of a command may be different for each command or may use the

default. The behavior when completing the command word itself may also be separately specified. These correspond to the following flags and arguments, all of which (except for -L) may be combined with any combination of the options described subsequently in the section `Option Flags':

command ...

controls completion for the named commands, which must be listed last on the com? mand line. If completion is attempted for a command with a pathname containing slashes and no completion definition is found, the search is retried with the last pathname component. If the command starts with a =, completion is tried with the pathname of the command.

Any of the command strings may be patterns of the form normally used for filename generation. These should be quoted to protect them from immediate expansion; for example the command string 'foo*' arranges for completion of the words of any com? mand beginning with foo. When completion is attempted, all pattern completions are tried in the reverse order of their definition until one matches. By default, com? pletion then proceeds as normal, i.e. the shell will try to generate more matches for the specific command on the command line; this can be overridden by including -tn in the flags for the pattern completion.

Note that aliases are expanded before the command name is determined unless the COMPLETE_ALIASES option is set. Commands may not be combined with the -C, -D or -T flags.

-C controls completion when the command word itself is being completed. If no compctl -C command has been issued, the names of any executable command (whether in the path or specific to the shell, such as aliases or functions) are completed.

-D controls default completion behavior for the arguments of commands not assigned any special behavior. If no compctl -D command has been issued, filenames are com? pleted.

-T supplies completion flags to be used before any other processing is done, even be? fore processing for compctls defined for specific commands. This is especially useful when combined with extended completion (the -x flag, see the section `Ex? tended Completion' below). Using this flag you can define default behavior which will apply to all commands without exception, or you can alter the standard behav? ior for all commands. For example, if your access to the user database is too slow

and/or it contains too many users (so that completion after `~' is too slow to be usable), you can use

compctl -T -x 's[~] C[0,[^/]#]' -k friends -S/ -tn

to complete the strings in the array friends after a `~'. The C[...] argument is necessary so that this form of ~-completion is not tried after the directory name is finished.

-L lists the existing completion behavior in a manner suitable for putting into a start-up script; the existing behavior is not changed. Any combination of the above forms, or the -M flag (which must follow the -L flag), may be specified, oth? erwise all defined completions are listed. Any other flags supplied are ignored.

no argument

If no argument is given, compctl lists all defined completions in an abbreviated form; with a list of options, all completions with those flags set (not counting extended completion) are listed.

If the + flag is alone and followed immediately by the command list, the completion behav? ior for all the commands in the list is reset to the default. In other words, completion will subsequently use the options specified by the -D flag.

The form with -M as the first and only option defines global matching specifications (see zshcompwid). The match specifications given will be used for every completion attempt (only when using compctl, not with the new completion system) and are tried in the order in which they are defined until one generates at least one match. E.g.:

compctl -M '' 'm:{a-zA-Z}={A-Za-z}'

This will first try completion without any global match specifications (the empty string) and, if that generates no matches, will try case insensitive completion.

OPTION FLAGS

[-fcFBdeaRGovNAIOPZEnbjrzu/12]

[-k array] [-g globstring] [-s substring]

[-K function]

[-Q] [-P prefix] [-S suffix]

[-W file-prefix] [-H num pattern]

[-q] [-X explanation] [-Y explanation]

[-y func-or-var] [-l cmd] [-h cmd] [-U]

[-t continue] [-J name] [-V name]

[-M match-spec]

The remaining options specify the type of command arguments to look for during completion.

Any combination of these flags may be specified; the result is a sorted list of all the possibilities. The options are as follows.

Simple Flags

These produce completion lists made up by the shell itself:

-f Filenames and file system paths.

-/ Just file system paths.

-c Command names, including aliases, shell functions, builtins and reserved words.

-F Function names.

-B Names of builtin commands.

-m Names of external commands.

-w Reserved words.

-a Alias names.

-R Names of regular (non-global) aliases.

-G Names of global aliases.

-d This can be combined with -F, -B, -w, -a, -R and -G to get names of disabled func? tions, builtins, reserved words or aliases.

-e This option (to show enabled commands) is in effect by default, but may be combined with -d; -de in combination with -F, -B, -w, -a, -R and -G will complete names of functions, builtins, reserved words or aliases whether or not they are disabled.

-o Names of shell options (see zshoptions(1)).

-v Names of any variable defined in the shell.

-N Names of scalar (non-array) parameters.

-A Array names.

-I Names of integer variables.

-O Names of read-only variables.

-p Names of parameters used by the shell (including special parameters).

-Z Names of shell special parameters.

-E Names of environment variables.

-n Named directories.

-b Key binding names.

-j Job names: the first word of the job leader's command line. This is useful with

the kill builtin.

-r Names of running jobs.

-z Names of suspended jobs.

-u User names.

Flags with Arguments

These have user supplied arguments to determine how the list of completions is to be made up:

-k array

Names taken from the elements of $array (note that the `$' does not appear on the command line). Alternatively, the argument array itself may be a set of space- or comma-separated values in parentheses, in which any delimiter may be escaped with a backslash; in this case the argument should be quoted. For example,

compctl -k "(cputime filesize datasize stacksize coredumpsize resident descriptors)" limit

-g globstring

The globstring is expanded using filename globbing; it should be quoted to protect it from immediate expansion. The resulting filenames are taken as the possible com? pletions. Use `*(/)' instead of `*/' for directories. The fignore special parame? ter is not applied to the resulting files. More than one pattern may be given sep? arated by blanks. (Note that brace expansion is not part of globbing. Use the syn? tax `(either|or)' to match alternatives.)

-s substring

The substring is split into words and these words are than expanded using all shell expansion mechanisms (see zshexpn(1)). The resulting words are taken as pos? sible completions. The fignore special parameter is not applied to the resulting files. Note that -g is faster for filenames.

-K function

Call the given function to get the completions. Unless the name starts with an un? derscore, the function is passed two arguments: the prefix and the suffix of the word on which completion is to be attempted, in other words those characters before the cursor position, and those from the cursor position onwards. The whole command line can be accessed with the -c and -l flags of the read builtin. The function should set the variable reply to an array containing the completions (one comple?

tion per element); note that reply should not be made local to the function. From such a function the command line can be accessed with the -c and -l flags to the read builtin. For example,

```
function whoson { reply=(`users`); }
compctl -K whoson talk
```

completes only logged-on users after `talk'. Note that `whoson' must return an ar? ray, so `reply=`users`' would be incorrect.

-H num pattern

The possible completions are taken from the last num history lines. Only words matching pattern are taken. If num is zero or negative the whole history is searched and if pattern is the empty string all words are taken (as with `*'). A typical use is

```
compctl -D -f + -H 0 ''
```

which forces completion to look back in the history list for a word if no filename matches.

Control Flags

These do not directly specify types of name to be completed, but manipulate the options that do:

-Q This instructs the shell not to quote any metacharacters in the possible comple? tions. Normally the results of a completion are inserted into the command line with any metacharacters quoted so that they are interpreted as normal characters. This is appropriate for filenames and ordinary strings. However, for special ef? fects, such as inserting a backquoted expression from a completion array (-k) so that the expression will not be evaluated until the complete line is executed, this option must be used.

-P prefix

The prefix is inserted just before the completed string; any initial part already typed will be completed and the whole prefix ignored for completion purposes. For example,

```
compctl -j -P "%" kill
```

inserts a `%' after the kill command and then completes job names.

-S suffix

When a completion is found the suffix is inserted after the completed string. In

the case of menu completion the suffix is inserted immediately, but it is still possible to cycle through the list of completions by repeatedly hitting the same key.

-W file-prefix

With directory file-prefix: for command, file, directory and globbing completion (options -c, -f, -/, -g), the file prefix is implicitly added in front of the com? pletion. For example,

compctl -/ -W ~/Mail maildirs

completes any subdirectories to any depth beneath the directory ~/Mail, although that prefix does not appear on the command line. The file-prefix may also be of the form accepted by the -k flag, i.e. the name of an array or a literal list in parenthesis. In this case all the directories in the list will be searched for pos? sible completions.

-q If used with a suffix as specified by the -S option, this causes the suffix to be removed if the next character typed is a blank or does not insert anything or if the suffix consists of only one character and the next character typed is the same character; this the same rule used for the AUTO_REMOVE_SLASH option. The option is most useful for list separators (comma, colon, etc.).

-l cmd This option restricts the range of command line words that are considered to be ar? guments. If combined with one of the extended completion patterns `p[...]', `r[...]', or `R[...]' (see the section `Extended Completion' below) the range is restricted to the range of arguments specified in the brackets. Completion is then performed as if these had been given as arguments to the cmd supplied with the op? tion. If the cmd string is empty the first word in the range is instead taken as the command name, and command name completion performed on the first word in the range. For example,

compctl -x 'r[-exec,;]' -l '' -- find

completes arguments between `-exec' and the following `;' (or the end of the com? mand line if there is no such string) as if they were a separate command line.

-h cmd Normally zsh completes quoted strings as a whole. With this option, completion can be done separately on different parts of such strings. It works like the -l option but makes the completion code work on the parts of the current word that are sepa? rated by spaces. These parts are completed as if they were arguments to the given

cmd. If cmd is the empty string, the first part is completed as a command name, as with -l.

-U Use the whole list of possible completions, whether or not they actually match the word on the command line. The word typed so far will be deleted. This is most useful with a function (given by the -K option) which can examine the word compo? nents passed to it (or via the read builtin's -c and -l flags) and use its own cri? teria to decide what matches. If there is no completion, the original word is re? tained. Since the produced possible completions seldom have interesting common prefixes and suffixes, menu completion is started immediately if AUTO_MENU is set and this flag is used.

-y func-or-var

The list provided by func-or-var is displayed instead of the list of completions whenever a listing is required; the actual completions to be inserted are not af? fected. It can be provided in two ways. Firstly, if func-or-var begins with a $ it defines a variable, or if it begins with a left parenthesis a literal array, which contains the list. A variable may have been set by a call to a function using the -K option. Otherwise it contains the name of a function which will be executed to create the list. The function will be passed as an argument list all matching com? pletions, including prefixes and suffixes expanded in full, and should set the ar? ray reply to the result. In both cases, the display list will only be retrieved after a complete list of matches has been created.

Note that the returned list does not have to correspond, even in length, to the original set of matches, and may be passed as a scalar instead of an array. No special formatting of characters is performed on the output in this case; in par? ticular, newlines are printed literally and if they appear output in columns is suppressed.

-X explanation

Print explanation when trying completion on the current set of options. A `%n' in this string is replaced by the number of matches that were added for this explana? tion string. The explanation only appears if completion was tried and there was no unique match, or when listing completions. Explanation strings will be listed to? gether with the matches of the group specified together with the -X option (using the -J or -V option). If the same explanation string is given to multiple -X op?

tions, the string appears only once (for each group) and the number of matches shown for the `%n' is the total number of all matches for each of these uses. In any case, the explanation string will only be shown if there was at least one match added for the explanation string.

The sequences %B, %b, %S, %s, %U, and %u specify output attributes (bold, standout, and underline), %F, %f, %K, %k specify foreground and background colours, and %{...%} can be used to include literal escape sequences as in prompts.

-Y explanation

Identical to -X, except that the explanation first undergoes expansion following the usual rules for strings in double quotes. The expansion will be carried out after any functions are called for the -K or -y options, allowing them to set vari? ables.

-t continue

The continue-string contains a character that specifies which set of completion flags should be used next. It is useful:

(i) With -T, or when trying a list of pattern completions, when compctl would usu? ally continue with ordinary processing after finding matches; this can be sup? pressed with `-tn'.

(ii) With a list of alternatives separated by +, when compctl would normally stop when one of the alternatives generates matches. It can be forced to consider the next set of completions by adding `-t+' to the flags of the alternative before the `+'.

(iii) In an extended completion list (see below), when compctl would normally con? tinue until a set of conditions succeeded, then use only the immediately following flags. With `-t-', compctl will continue trying extended completions after the next `-'; with `-tx' it will attempt completion with the default flags, in other words those before the `-x'.

-J name

This gives the name of the group the matches should be placed in. Groups are listed and sorted separately; likewise, menu completion will offer the matches in the groups in the order in which the groups were defined. If no group name is explic? itly given, the matches are stored in a group named default. The first time a group name is encountered, a group with that name is created. After that all matches with

the same group name are stored in that group.

This can be useful with non-exclusive alternative completions. For example, in

compctl -f -J files -t+ + -v -J variables foo

both files and variables are possible completions, as the -t+ forces both sets of alternatives before and after the + to be considered at once. Because of the -J options, however, all files are listed before all variables.

-V name

Like -J, but matches within the group will not be sorted in listings nor in menu completion. These unsorted groups are in a different name space from the sorted ones, so groups defined as -J files and -V files are distinct.

-1 If given together with the -V option, makes only consecutive duplicates in the group be removed. Note that groups with and without this flag are in different name spaces.

-2 If given together with the -J or -V option, makes all duplicates be kept. Again, groups with and without this flag are in different name spaces.

-M match-spec

This defines additional matching control specifications that should be used only when testing words for the list of flags this flag appears in. The format of the match-spec string is described in zshcompwid.

ALTERNATIVE COMPLETION

compctl [-CDT] options + options [+ ...] [+] command ...

The form with `+' specifies alternative options. Completion is tried with the options be? fore the first `+'. If this produces no matches completion is tried with the flags after the `+' and so on. If there are no flags after the last `+' and a match has not been found up to that point, default completion is tried. If the list of flags contains a -t with a + character, the next list of flags is used even if the current list produced matches.

Additional options are available that restrict completion to some part of the command line; this is referred to as `extended completion'.

EXTENDED COMPLETION

compctl [-CDT] options -x pattern options - ... --

[command ...]

compctl [-CDT] options [-x pattern options - ... --]

[+ options [-x ... --] ... [+]] [command ...]

The form with `-x' specifies extended completion for the commands given; as shown, it may be combined with alternative completion using `+'. Each pattern is examined in turn; when a match is found, the corresponding options, as described in the section `Option Flags' above, are used to generate possible completions. If no pattern matches, the options given before the -x are used.

Note that each pattern should be supplied as a single argument and should be quoted to prevent expansion of metacharacters by the shell.

A pattern is built of sub-patterns separated by commas; it matches if at least one of these sub-patterns matches (they are `or'ed). These sub-patterns are in turn composed of other sub-patterns separated by white spaces which match if all of the sub-patterns match (they are `and'ed). An element of the sub-patterns is of the form `c[...][...]', where the pairs of brackets may be repeated as often as necessary, and matches if any of the sets of brackets match (an `or'). The example below makes this clearer.

The elements may be any of the following:

s[string]...

Matches if the current word on the command line starts with one of the strings given in brackets. The string is not removed and is not part of the completion.

S[string]...

Like s[string] except that the string is part of the completion.

p[from,to]...

Matches if the number of the current word is between one of the from and to pairs inclusive. The comma and to are optional; to defaults to the same value as from. The numbers may be negative: -n refers to the n'th last word on the line.

c[offset,string]...

Matches if the string matches the word offset by offset from the current word posi? tion. Usually offset will be negative.

C[offset,pattern]...

Like c but using pattern matching instead.

w[index,string]...

Matches if the word in position index is equal to the corresponding string. Note that the word count is made after any alias expansion.

W[index,pattern]...

Like w but using pattern matching instead.

n[index,string]...

Matches if the current word contains string. Anything up to and including the in?

dexth occurrence of this string will not be considered part of the completion, but

the rest will. index may be negative to count from the end: in most cases, index

will be 1 or -1. For example,

compctl -s '`users`' -x 'n[1,@]' -k hosts -- talk

will usually complete usernames, but if you insert an @ after the name, names from

the array hosts (assumed to contain hostnames, though you must make the array your?

self) will be completed. Other commands such as rcp can be handled similarly.

N[index,string]...

Like n except that the string will be taken as a character class. Anything up to

and including the indexth occurrence of any of the characters in string will not be

considered part of the completion.

m[min,max]...

Matches if the total number of words lies between min and max inclusive.

r[str1,str2]...

Matches if the cursor is after a word with prefix str1. If there is also a word

with prefix str2 on the command line after the one matched by str1 it matches only

if the cursor is before this word. If the comma and str2 are omitted, it matches if

the cursor is after a word with prefix str1.

R[str1,str2]...

Like r but using pattern matching instead.

q[str]...

Matches the word currently being completed is in single quotes and the str begins

with the letter `s', or if completion is done in double quotes and str starts with

the letter `d', or if completion is done in backticks and str starts with a `b'.

EXAMPLE

compctl -u -x 's[+] c[-1,-f],s[-f+]' \

-g '~/Mail/*(:t)' - 's[-f],c[-1,-f]' -f -- mail

This is to be interpreted as follows:

If the current command is mail, then

if ((the current word begins with + and the previous word is -f)

or (the current word begins with -f+)), then complete the

non-directory part (the `:t' glob modifier) of files in the directory

~/Mail; else

if the current word begins with -f or the previous word was -f, then

complete any file; else

complete user names.