NAME

zshroadmap − informal introduction to the zsh manual The Zsh Manual, like the shell itself, is large and often complicated. This section of the manual provides some pointers to areas of the shell that are likely to be of particular interest to new users, and indicates where in the rest of the manual the documentation is to be found.

WHEN THE SHELL STARTS

When it starts, the shell reads commands from various files. These can be created or edited to customize the shell. See the section Startup/Shutdown Files in *zsh*(1).

If no personal initialization files exist for the current user, a function is run to help you change some of the most common settings. It won't appear if your administrator has disabled the **zsh/newuser** module. The function is designed to be self−explanatory. You can run it by hand with '**autoload −Uz zsh−newuser−install; zsh−newuser−install −f**'. See also the section User Configuration Functions in *zshcontrib*(1).

INTERACTIVE USE

Interaction with the shell uses the builtin Zsh Line Editor, ZLE. This is described in detail in *zshzle*(1).

The first decision a user must make is whether to use the Emacs or Vi editing mode as the keys for editing are substantially different. Emacs editing mode is probably more natural for beginners and can be selected explicitly with the command **bindkey −e**.

A history mechanism for retrieving previously typed lines (most simply with the Up or Down arrow keys) is available; note that, unlike other shells, zsh will not save these lines when the shell exits unless you set appropriate variables, and the number of history lines retained by default is quite small (30 lines). See the description of the shell variables (referred to in the documentation as parameters) **HISTFILE**, **HISTSIZE** and **SAVEHIST** in *zshparam*(1). Note that it's currently only possible to read and write files saving history when the shell is interactive, i.e. it does not work from scripts.

The shell now supports the UTF−8 character set (and also others if supported by the operating system). This is (mostly) handled transparently by the shell, but the degree of support in terminal emulators is variable. There is some discussion of this in the shell FAQ, **http://www.zsh.org/FAQ/**. Note in particular that for combining characters to be handled the option **COMBINING_CHARS** needs to be set. Because the shell is now more sensitive to the definition of the character set, note that if you are upgrading from an older version of the shell you should ensure that the appropriate variable, either **LANG** (to affect all aspects of the shell's operation) or **LC_CTYPE** (to affect only the handling of character sets) is set to an appropriate value. This is true even if you are using a single−byte character set including extensions of ASCII such as **ISO−8859−1** or **ISO−8859−15**. See the description of **LC_CTYPE** in *zshparam*(1).

Completion

Completion is a feature present in many shells. It allows the user to type only a part (usually the prefix) of a word and have the shell fill in the rest. The completion system in zsh is programmable. For example, the shell can be set to complete email addresses in arguments to the mail command from your **~/.abook/addressbook**; usernames, hostnames, and even remote paths in arguments to scp, and so on. Anything that can be written in or glued together with zsh can be the source of what the line editor offers as possible completions.

Zsh has two completion systems, an old, so called **compctl** completion (named after the builtin command that serves as its complete and only user interface), and a new one, referred to as **compsys**, organized as library of builtin and user−defined functions. The two systems differ in their interface for specifying the completion behavior. The new system is more customizable and is supplied with completions for many commonly used commands; it is therefore to be preferred.

The completion system must be enabled explicitly when the shell starts. For more information see *zshcompsys*(1).

Extending the line editor

Apart from completion, the line editor is highly extensible by means of shell functions. Some useful functions are provided with the shell; they provide facilities such as:

insert−composed−char
composing characters not found on the keyboard

match−words−by−style
configuring what the line editor considers a word when moving or deleting by word

history−beginning−search−backward−end, etc.
alternative ways of searching the shell history

replace−string, **replace−pattern**
functions for replacing strings or patterns globally in the command line

edit−command−line
edit the command line with an external editor.

See the section 'ZLE Functions' in *zshcontrib*(1) for descriptions of these.

OPTIONS

The shell has a large number of options for changing its behaviour. These cover all aspects of the shell; browsing the full documentation is the only good way to become acquainted with the many possibilities. See *zshoptions*(1).

PATTERN MATCHING

The shell has a rich set of patterns which are available for file matching (described in the documentation as 'filename generation' and also known for historical reasons as 'globbing') and for use when programming. These are described in the section 'Filename Generation' in *zshexpn*(1).

Of particular interest are the following patterns that are not commonly supported by other systems of pattern matching:

****** for matching over multiple directories

| for matching either of two alternatives

~, **^** the ability to exclude patterns from matching when the **EXTENDED_GLOB** option is set

(...) glob qualifiers, included in parentheses at the end of the pattern, which select files by type (such as directories) or attribute (such as size).

GENERAL COMMENTS ON SYNTAX

Although the syntax of zsh is in ways similar to the Korn shell, and therefore more remotely to the original UNIX shell, the Bourne shell, its default behaviour does not entirely correspond to those shells. General shell syntax is introduced in the section 'Shell Grammar' in *zshmisc*(1).

One commonly encountered difference is that variables substituted onto the command line are not split into words. See the description of the shell option **SH_WORD_SPLIT** in the section 'Parameter Expansion' in *zshexpn*(1). In zsh, you can either explicitly request the splitting (e.g. **${=foo}**) or use an array when you want a variable to expand to more than one word. See the section 'Array Parameters' in *zshparam*(1).

PROGRAMMING

The most convenient way of adding enhancements to the shell is typically by writing a shell function and arranging for it to be autoloaded. Functions are described in the section 'Functions' in *zshmisc*(1). Users changing from the C shell and its relatives should notice that aliases are less used in zsh as they don't perform argument substitution, only simple text replacement.

A few general functions, other than those for the line editor described above, are provided with the shell and are described in *zshcontrib*(1). Features include:

promptinit
a prompt theme system for changing prompts easily, see the section 'Prompt Themes'

zsh−mime−setup
a MIME−handling system which dispatches commands according to the suffix of a file as done by graphical file managers

zcalc a calculator

zargs a version of **xargs** that makes the **find** command redundant

zmv a command for renaming files by means of shell patterns.