zsh: Power Tools. Because having all your fingers is overrated.

Paul L. Snyder <plsnyder@drexel.edu>

January 12, 2009

zsh: Power Tools. Because having all your fingers is overrated.

* The original Bourne shell
* Written by Stephen Bourne for Version 7 Unix in 1977
* Usually, your '/bin/sh' is another shell running in compatibility mode

Why should I use it?
* You are writing a "portable" shell script
* Your life will still be hell
* If you're writing a large system, use a real programming lanugage (Perl, Python, etc.)

* David Korn's Korn shell
* MUCH better for programming than sh or *shudder* csh
* Some interactive improvements
* Two major versions, ksh88 and ksh93 (finally open-sourced in 2000)
* The FSF released pdksh (mostly ksh88-compatible)

Why should I use it?

* Misguided nostalgia

```
* The "C" shell
* Ha!
* Introduced many now-standard interactive
  features (job control, aliases, !-substitution
* Nastily brain-damaged scripting behavior
* Csh Programming Considered Harmful:
http://www.faqs.org/faqs/unix-faq/shell/csh-whynot
* Implementations were historically very buggy
* The modern (and proprietary) Hamilton C Shell bri
  the glory of csh to Windows.

Why should I use it?
* You are an idiot.
```

* The 't' is for 'TENEX'
* The TENEX OS (later TOPS-20) had command completion facilities
* tcsh introduced programmable command completion to Unix shells
* Kept the csh syntax

Why should I use it?

* You like having cool people laugh at you.
* Actually, there is no good reason to use tcsh.

* The Bourne-again shell
* ksh was proprietary and csh sucked, so bash was created
* Considered a ksh descendant
* Default shell for most Linux distributions
* /bin/sh is usually bash under the covers (except on Ubuntuish systems)

Why should I use it?

* You like the safety of being one of the herd
* Power scares you
* You are pathetically grateful for finally receiving features zsh has had for years

* Mostly descended from ksh
* Absorbs interesting (and possibly conflicting) features from other shells
* Possesses the lucid clarity of perl...
* ...and the streamlined elegance of emacs

Why should I use it?

* See remainder of presentation, below

* Scripting
  - Make common tasks easier
  - Use POSIX sh for portability, not zsh
    = Portable shell code is non-trivial
  - Don't write large systems using shell scripting!
    = Use, e.g., Perl, Python, Ruby

* Interactive use
  - Make common tasks easier
  - Flexible command-line editing and history
  - Globbing
  - Completion

* Startup files
* zsh's modular design
* Variables
* Expansion and substitution
* Interactive use
* Completion

* zsh in all its gory glory is unspeakably complex

* Happily, you don't need to know much to start using the shell

* zsh rewards knowledge with power

Login files are run in this order ($ZDOTDIR defaults to $HOME):

/etc/zshenv	All shells, can't be overridden
$ZDOTDIR/.zshenv	All shells (with RCS option)
/etc/zprofile	Login shells (with GLOBAL_RCS)
$ZDOTDIR/.zprofile	Login shells (with RCS)
/etc/zshrc	Interactive shells (GLOBAL_RCS)
$ZDOTDIR/.zshrc	Interactive shells (with RCS)
/etc/zlogin	Login shells (with GLOBAL_RCS)
$ZDOTDIR/.zlogin	Login shells (with RCS)

When exiting:

```
$ZDOTDIR/.zlogout   Login shells (with RCS option)
/etc/zlogout        Login shells (with GLOBAL_RCS)
```

* zsh is pretty bland until it has been configured.
* Most of the cool options are turned off by default.
* Two courses:

  - Steal someone else' .zshrc

  - Use the menu-based zsh-newuser-install (many distributions configure this to be run by default)

- If this isn't the case you can run it yourself:

```
$ autoload -Uz zsh-newuser-install;
zsh-newuser-install -f
```

* Portions of the shell are compiled as optional .so modules. They can be loaded using 'zmodload' Modules include:
  - zsh/zftp: command-line FTP program
  - zsh/complete: Programmable completion system
  - zsh/net/socket: 'zsocket' command to maniplate UNIX domain sockets
  - zsh/net/tcp: 'ztcp' command to create and accept TCP connections
  - zsh/zpty: Run a command under its own pseudo-terminal

* A few more modules of interest:
  - zsh/termcap and zsh/terminfo: output termcap and terminfo sequences by capability name
  - zsh/mapfile: tie a file to an associative array
  - zsh/newuser: menu-drive dot-file creation for new users
  - zsh/pcre and zsh/regex: Perl-compatible and POSIX regexes

```
$ beer=('Hop Devil' 'Golden Monkey' \
> 'Old Horizontal')
$ print $beer[2]    # or ${beer[2]}
Golden Monkey
$ print $foo[-1]    # negative subscripts allowed
Old Horizontal
```

* Note that zsh arrays start from 1, not 0!
* 'setopt ksharrays' for ksh-style behavior
* bash 3 supports arrays of this type

```
$ typeset -A collective
$ collective=(larks exaltation ravens \
> unkindness crows murder)
$ print $collective[larks]
exaltation
$ print ${(k)collective}
larks ravens crows
```

* You may be familiar with these as perl hashes
* Associative array support will be coming in bash 4

```
* Integers:                          typeset -i foo
* Alternate base integers:           typeset -i 16 bar
* Floating point, fixed notation:    typeset -F baz
* Floating point, sci notation:      typeset -E womble

$ zmodload zsh/mathfunc
$ (( pi = 4.0 * atan(1.0) ))
$ echo $pi
3.1415926536
```

* Create a tied variable/array pair: `typeset -T FOO foo`

* Create a variable that always expands to lowercase: `typeset -l BAR`

* Or uppercase: `typeset -u BAZ`

* Make a variable read-only: `typeset -r WOMBLE`

* Keep array entries unique: `typeset -U path`

* PS1 displayed at regular command prompt
* PS2 for second-level prompt
  - Also displays details of nested shell constructs
* PS3 displayed inside 'select' construct'
* PS4 is the trace prompt
* RPS1 and RPS2 are right prompts! 'RPS1=%t'

```
%M  FQ hostname                    %#  '#' if shell has
                                   root, '%' otherwise
%m  hostname up to '.'             %_  nesting status of
                                   shell constructs (PS2)
%n  username                       %d  Present working
                                   directory - $PWD
%y  User's login tty               %3d Last three
                                   components of $PWD
%h  Current history num            %i  Line number of a
                                   trace (for PS4)
%n  Current script or func         %D  Date in yy-mm-dd
%t time in 12-hour format
```

```
%{..%} Escape sequence      %D{format} format
                            date using strftime(3)
%(x.true-text.false-text)   Ternary expression
                            E.g. %# = %(!.#.$)
Alternately %n(x.true-text.false-text)
         or %(nx.true-text.false-text)
         where n is an integer

Tests:
  ! privileges  # effective uid is n
  ? exit status of last command is n
  d day of the month is n
  / current absolute path has n elements
```

When you enter a command at the prompt, it is mangled as follows:

1. History Expansion
2. Alias Expansion
3. Process Substitution,
   Parameter Expansion,
   Command Substitution,
   Arithmetic Substitution, and
   Brace Expansion
4. Filename Expansion
5. Filename Generation ("globbing")

* History is inspired by csh's history system
  - setopt CSH_JUNKIE_HISTORY to lobotomize zsh

!! is the last command executed
!!$ is the last word of the last command
!n refers to history command numbered 'n'
'history' for a list, or add '%h' to your prompt
!str last command starting with 'str'
!# is the command you are typing right now!
!?str[?] is the last command containing 'str'
!{...} prevents confusion with surrounding text

```
0    the first word
n    the nth argument
^    the first argument
$    the last argument
%    the word match by a 'str' search
n-m  words n through m
*    all the arguments
```

For extra fun, use these with regular parameters!

h	remove one trailing path component
t	remove all but the last path component
r	remove filename extension
e	remove everything but the extension
l	convert all words to lowercase
u	convert all words to uppercase

[g]s/old/new[/] Replace 'old' string with 'new'.
if 'new' contains '&', '&' is replaced with 'old'

```
f        repeat following modifier exhaustively
F:expr:  repeat following modifier expr times
w        apply following modifier to each word
W:sep:   like w, but applies to parts of string
           that are separated by 'sep'
```

```
* All the usual suspects. E.g.:
$ echo ${foo?BAR}
BAR
$ foo=FOO
$ echo ${foo?BAR}
FOO
$ baz=/this/is/a/path
$ echo ${baz%is*}
/this/
$ echo ${baz/*is}
/a/path
```

```
There are LOTS of these...this is just a small
 selection
Place in parentheses before the parameter name,
 e.g., ${(%)PS1}

 %  Expand prompt sequences
 C  Capitalize each resulting word
 L  Convert all letters to lowercase
 o  sort words in ascending order
 O  sort words in descending order
 u  expand only first unique occurence
     of each word
 j:str: join the words of arrays using 'str'
 q  quote the expanded words
```

- You all know about *, ?, [x] and [^y].
- How about 'ls bar<2-6>'? Only matches existing files
- ^*FOO* globs all files without 'FOO' in their names
- *(foo|bar)* globs files with either 'foo' or 'bar'
- ba^z* globs 'bar' but not 'baz'
- (foo)# matches zero or more 'foo's...(foo)## matches any number

- Use ksh-style glob operators to tweak your parentheses
- *(foo) matches zero or more 'foo's
- ?(foo) matches zero or one 'foo's
- +(foo) matches one of more 'foo's
- !(foo) match anything BUT 'foo'
- Are you frightened yet?

- (#l) lowercase characters match upper or lower case; uppercase matches uppercase
- (#I) reenables case sensitivity
- (#b) activate backreferences for parenthesized groups; store the matches in the $match arr and the indices in $mbegin and $mend
- (#B) ends backreferencing.

- (#aN) Use approximate matching! Allow up to N errors in the match.

Glob qualifiers appear in parentheses at the end of a glob specifier...

```
*(.) matches regular files only
*(/) matches all directories
*(@) matches all symbolic links
*(x) matches all owner-executable files
*(s) matches all setuid files
*(f{go+w}) matches group or other-writeable
 files!
```

This next one even scares me...

```
*(estring) executes string as shell code!  The
   currently matched file is available in $REPLY;
   override the return with $reply or $REPLY.
```

Yipes!

One last trick...

```
$ ls **/foo*
```

Matches 'foo*' in current directory or any subdir

```
for x in *; do mv $x ${x:r}.bak; done
```

Too much work! In zsh, just use

```
for x in *; mv $x ${x:r}.bak
```

Actually, this is now deprecated, so it's a bad habit that I keep using it.

Similar short forms exist for 'if', 'while', and so on.

Even better better:

```
zmv '(file0?)' '$1.bak'
```

* zmv is the command-line rename tool you've always wanted

```
$ zmv '(*)-(*).mpeg3' '$2_$1.mp3'
$ zmv '(*)' '${(L)1}'
$ alias mmv='noglob zmv -W'
$ mmv *.pl.bak backups/*.pl
```

* Oopsing commands? 'setopt CORRECT'
* Fat-fingering filenames? 'setopt CORRECT_ALL'
* Prevent a command from being corrected:
  alias mv='nocorrect mv'

setopt MULTIOS for built-in tee functionality

```
$ ls >>file1 >file2 | cat
```

```
$ : > *
```

This truncates every file matched by *!

Well, not quite, as long as NO_CLOBBER is set.
For maximum damage, use

```
$ : >| *
```

How about a NULLCMD?

```
$ <first <second >combined
```

Change the default command by setting NULLCMD to something other than 'cat'

```
$ >combined
zsh: file exists: combined
```

NOCLOBBER saves your bacon.

* Multi-line editing

* Variable editing! 'vared path'

* 'zed' is zsh's built-in editor...use your zsh bindings for a quick edit
  (use C-j to save and exit or C-g to cancel)

* One of my favorites: the buffer stack

  $ bindkey '\eq' push-line-or-edit

* Stuff the buffer with 'print -z'

* I'm not going to tell you how to write new completion functions
* I don't want to be lynched.
* Besides, most anything you'd want to complete is probably in there already.
* To get started using completion, just turn it on when you run zsh-newuser-install

* Forgot an option? Just hit TAB

* Look at completions for 'tar', 'mplayer', 'emerge', or 'dpkg'

* man zsh and its 15 subpages (or just man zshall)

* http://www.zsh.org

  - Read the zsh user manual...friendly and useful

* Tips and tricks

  - http://www.rayninfo.co.uk/tips/zshtips.html
  - http://www.grml.org/zsh/zsh-lovers.html

* http://www.zshwiki.org

zsh: Power Tools. Because having all your fingers is overrated.

└Fin.

```
$ files=(*); echo $files[$RANDOM%$#files]]
```

```
$ alias -s txt=less
```

This can be bad for security!

```
$ alias -g ...='../..'
$ alias -g L='| less'
$ alias -g G='| egrep'
```