

ZSH

A better Shell for Interactive Use

Zuo, Le

China Systems and Technology Laboratory (CSTL), IBM

December 8, 2011

Outline

- 1 Why another shell?
- 2 New changes to common shell features
 - A completion system that rocks
 - aliasing
 - alternative statement structure
 - hook functions
 - globbing
- 3 ZLE widgets
- 4 Program your own completion function

Outline

1 Why another shell?

2 New changes to common shell features

- A completion system that rocks
- aliasing
- alternative statement structure
- hook functions
- globbing

3 ZLE widgets

4 Program your own completion function

Why zsh

A shell can be used in two manners:

in scripts shell is used to serialize a bunch of command for batch procession. For better compatibility, sh or bash are most frequently used.

interactively shell is used to interact with user in real time. Most common ones are: bash, ksh (AIX), tcsh (Solaris), fish, zsh.

Zsh is a shell designed for interactive use, although it is also a powerful scripting language. (<http://www.zsh.org>)

Why zsh

A shell can be used in two manners:

in scripts shell is used to serialize a bunch of command for batch procession. For better compatibility, sh or bash are most frequently used.

interactively shell is used to interact with user in real time. Most common ones are: bash, ksh (AIX), tcsh (Solaris), fish, zsh.

Zsh is a shell designed for interactive use, although it is also a powerful scripting language. (<http://www.zsh.org>)

Why zsh

A shell can be used in two manners:

in scripts shell is used to serialize a bunch of command for batch procession. For better compatibility, sh or bash are most frequently used.

interactively shell is used to interact with user in real time. Most common ones are: bash, ksh (AIX), tcsh (Solaris), fish, zsh.

Zsh is a shell designed for interactive use, although it is also a powerful scripting language. (<http://www.zsh.org>)

Pros and Cons

Pros

- You type less, and with the same amount of effort, the computer does more
- You get fancier response from computer

Cons

- You get addicted
- You feel a little bit “not at home” when having to use bash (I use zsh even on AIX !)

Pros and Cons

Pros

- You type less, and with the same amount of effort, the computer does more
- You get fancier response from computer

Cons

- You get addicted
- You feel a little bit “not at home” when having to use bash (I use zsh even on AIX !)

Attention

- zsh is very feature rich. This section only covers the features that the author has used so far.
- This section is to be a quick show case of zsh features. Some configurational details are omitted.

My zsh configuration file

```
https://github.com/roylez/dotfiles/blob/  
master/.zshrc
```

Attention

- zsh is very feature rich. This section only covers the features that the author has used so far.
- This section is to be a quick show case of zsh features. Some configurational details are omitted.

My zsh configuration file

```
https://github.com/roylez/dotfiles/blob/  
master/.zshrc
```

Outline

1 Why another shell?

2 New changes to common shell features

- A completion system that rocks
- aliasing
- alternative statement structure
- hook functions
- globbing

3 ZLE widgets

4 Program your own completion function

Outline

- 1 Why another shell?
- 2 New changes to common shell features
 - A completion system that rocks
 - aliasing
 - alternative statement structure
 - hook functions
 - globbing
- 3 ZLE widgets
- 4 Program your own completion function

Problems with bash completion system

- It is not intuitive enough.
- Inconvenient when traversing completion options.
- Its coverage of commands is limited.

Show-case: `ls`, `kill`, `ssh`

- `ls`
- `kill`
- `ssh`

Show-case: `cd -`

In `.zshrc`

```
setopt auto_pushd  
setopt pushd_ignore_dups  
setopt pushd_silent
```

```
roylez@bender> cd workspace/ruby
```

```
roylez@bender> cd ../test
```

```
roylez@bender> cd -
```

```
== directory stack ==
```

```
0 -- /home/roylez/workspace/supervise
```

```
1 -- /home/roylez/workspace
```

Outline

- 1 Why another shell?
- 2 New changes to common shell features
 - A completion system that rocks
 - **aliasing**
 - alternative statement structure
 - hook functions
 - globbing
- 3 ZLE widgets
- 4 Program your own completion function

Normal aliases

In .zshrc

```
alias vi='vim'
alias df='df -Th'
alias rsync='rsync --progress --partial'
alias ls='$ls -h --color=auto -X'
alias gfw="ssh -C2g -c arcfour \  
-o ServerAliveInterval=60 -D 7070"
```

Global aliases

Definition

If the `-g` flag is present, define a global alias; global aliases are expanded even if they do not occur in command position. (`man zshall`)

In `.zshrc`

```
alias -g A="|awk"  
alias -g L="|less"  
alias -g R="|tac"  
alias -g C="|wc"
```

Global aliases (cont'd)

In .zshrc

```
alias -g N="> /dev/null"

alias -g H="|head -n $(( $LINES-2 ))"

alias -g G='|GREP_COLOR=$(echo 3$[$(date \
    +%N)%6+1]'\;" ;1;7'\") egrep -i --color=always'

alias -g B='|sed -r "s:\x1B\[ [0-9;]*[mK]::g"'
```

Global aliases (cont'd)

In .zshrc

```
alias -g N="> /dev/null"
```

```
alias -g H="|head -n $(( $LINES-2 ))"
```

```
alias -g G='|GREP_COLOR=$(echo 3$[$(date \\\n +%N)%6+1]'\'';1;7'\''") egrep -i --color=always'
```

```
alias -g B='|sed -r "s:\x1B\[ [0-9;]*[mK]::g"'
```

Global aliases (cont'd)

In .zshrc

```
alias -g N="> /dev/null"

alias -g H="|head -n $(( $LINES-2 ))"

alias -g G='|GREP_COLOR=$(echo 3$[$(date \
    +%N)%6+1]'\;" ;1;7'\") egrep -i --color=always'

alias -g B='|sed -r "s:\x1B\[ [0-9;]*[mK]::g"'
```

Global aliases (cont'd)

In .zshrc

```
alias -g N="> /dev/null"

alias -g H="|head -n $(( $LINES-2 ))"

alias -g G='|GREP_COLOR=$(echo 3$[$(date \
    +%N)%6+1]'\;" ;1;7'\") egrep -i --color=always'

alias -g B='|sed -r "s:\x1B\[ [0-9;]*[mK]::g"'
```

Suffix aliases

Definition

If the `-s` flag is present, define a suffix alias: if the command word on a command line is in the form `'text.name'`, where `text` is any non-empty string, it is replaced by the text `'value text.name'`. (man `zshall`)

in `.zshrc`

```
alias -s ps=gv
for i in jpg png; alias -s $i=gqview
for i in avi rmvb wmv; alias -s $i=mplayer
for i in rar zip 7z lzma; alias -s $i="7z x"
```

Outline

- 1 Why another shell?
- 2 New changes to common shell features
 - A completion system that rocks
 - aliasing
 - **alternative statement structure**
 - hook functions
 - globbing
- 3 ZLE widgets
- 4 Program your own completion function

for

sh

```
for i in list; do sublist ; done  
# for i in a b c d; do echo $i; done
```

zsh alternative

```
for i in list; sublist  
# for i in a b c d; echo $i  
for i ( list ) sublist  
# for i ( {a-d} ) echo $i
```

for

sh

```
for i in list; do sublist ; done
# for i in a b c d; do echo $i; done
```

zsh alternative

```
for i in list; sublist
# for i in a b c d; echo $i
for i ( list ) sublist
# for i ( {a-d} ) echo $i
```

while

sh

```
while list-1; do list-2; done
```

zsh alternative

```
while list-1 { list-2 }
```

while

sh

```
while list-1; do list-2; done
```

zsh alternative

```
while list-1 { list-2 }
```

if

sh

```
if list-1; then list-2; fi  
# if [[ $HOME = $PWD ]]; then echo at home; fi
```

zsh alternative

```
if list-1 list-2  
# if [[ $HOME = $PWD ]] echo at home
```

if

sh

```
if list-1; then list-2; fi  
# if [[ $HOME = $PWD ]]; then echo at home; fi
```

zsh alternative

```
if list-1 list-2  
# if [[ $HOME = $PWD ]] echo at home
```

Outline

- 1 Why another shell?
- 2 New changes to common shell features
 - A completion system that rocks
 - aliasing
 - alternative statement structure
 - **hook functions**
 - globbing
- 3 ZLE widgets
- 4 Program your own completion function

hooks

chpwd Executed whenever the current working directory is changed.

precmd Executed before each prompt.

preexec Executed just after a command has been read and is about to be executed.

defining hooks

in .zshrc

```
function precmd() {  
    screen_precmd  
    git_branch_precmd  
}  
function preexec() {  
    screen_preexec  
    pwd_color_preexec  
}  
function chpwd() {  
    pwd_color_chpwd  
    git_branch_chpwd  
}
```

Sample applications of hooks

- Dynamically set GNU screen/xterm title

```
0 irssi 1 gfw 2 ~ 3 ~ 4 ~ 5 vim 6 man 7 import
```

- Dynamically change prompt

```
roylez@bender> pwd                                ~ 22:04:46
/home/roylez
roylez@bender> j octo                               ~ 22:04:53
/home/roylez/workspace/ruby/octopress
roylez@bender master > ~/workspace/ruby/octopress 22:05:02
roylez@bender master > echo aa >> Gemfile
roylez@bender master > git s ~/workspace/ruby/octopress 22:05:22
M Gemfile
roylez@bender master* > ~/workspace/ruby/octopress 22:05:24
```

- Change shell environment variables according to directory change and etc

Outline

- 1 Why another shell?
- 2 New changes to common shell features
 - A completion system that rocks
 - aliasing
 - alternative statement structure
 - hook functions
 - globbing
- 3 ZLE widgets
- 4 Program your own completion function

globbing that we all know...

- * matches anything
- ? matches any single character
- [abc] matches characters listed in brackets

zsh's globbing has...

globbing flags

`(#X)foo` where X is the flag

matches Gemfile as well!

`(#i)gemfile`

zsh's globbing has...

recursive globbing

`(foo/)#` matches string that contains zero or more foo in the middle

zsh's globbing has...

glob qualifiers

`foo(XYZ)` qualifiers XYZ makes foo pattern more specific

```
# NF global alias always points  
# to the newest file/dir  
alias -g NF='./*(oc[1])'
```

Outline

- 1 Why another shell?
- 2 New changes to common shell features
 - A completion system that rocks
 - aliasing
 - alternative statement structure
 - hook functions
 - globbing
- 3 ZLE widgets
- 4 Program your own completion function

ZLE?

ZLE is

- zsh command line editor
- to provide similar to but better than functions that `readline` provides
- easy to program

Command Line Editor???

A command line editor:

- helps defining how the cursor moves, how text is input/deleted, in command line
- governs all aspect of editing when you have a terminal connection

readline:

- is the most widely used command line editor library (but zsh says they have a better choice)
- its run time configuration file is `.inputrc` (this file affects all programmes that relies on readline, including `iPython`, `GNUplot`, `irb`, `bash`, `octave` and a lot more)

Command Line Editor???

A command line editor:

- helps defining how the cursor moves, how text is input/deleted, in command line
- governs all aspect of editing when you have a terminal connection

readline:

- is the most widely used command line editor library (but zsh says they have a better choice)
- its run time configuration file is `.inputrc` (this file affects all programmes that relies on readline, including `iPython`, `GNUplot`, `irb`, `bash`, `octave` and a lot more)

ZLE widget: sudo-command-line

```
sudo-command-line() {  
    [[ -z $BUFFER ]] && zle up-history  
    [[ $BUFFER != sudo\ * ]] && \  
        BUFFER="sudo $BUFFER"  
    zle end-of-line # move cursor to EOL  
}  
zle -N sudo-command-line  
# define shortcut: [Esc] [Esc]  
bindkey "\e\e" sudo-command-line
```

ZLE widget: dumb-cd

```
# from linux-toy.org
dumb-cd() {
    if [[ -n $BUFFER ]] ; then
        zle expand-or-complete
    else
        BUFFER="cd "
        zle end-of-line
        zle expand-or-complete
    fi
}
zle -N dumb-cd
bindkey "\t" dumb-cd
```

Outline

- 1 Why another shell?
- 2 New changes to common shell features
 - A completion system that rocks
 - aliasing
 - alternative statement structure
 - hook functions
 - globbing
- 3 ZLE widgets
- 4 Program your own completion function

Attention

- zsh completion system is very feature rich and complex.
- documentation can be found in `zshcompsys` and `zshcompwid` manpages.
- best way to learn programming completion function is still to read the ones shipped with zsh official build.

Attention

- zsh completion system is very feature rich and complex.
- documentation can be found in [zshcompsys](#) and [zshcompwid](#) manpages.
- best way to learn programming completion function is still to read the ones shipped with zsh official build.

Attention

- zsh completion system is very feature rich and complex.
- documentation can be found in [zshcompsys](#) and [zshcompwid](#) manpages.
- best way to learn programming completion function is still to read the ones shipped with zsh official build.

Completion function file

A completion function file:

- normally named `_foo` if the targeting command to be completed is `foo`
- must be in zsh's function path `$fpath`
- starts with a line like `"#compdef foo"`

Completion function file

A completion function file:

- normally named `_foo` if the targeting command to be completed is `foo`
- must be in zsh's function path `$fpath`
- starts with a line like `#compdef foo`

Completion function file

A completion function file:

- normally named `_foo` if the targeting command to be completed is `foo`
- must be in zsh's function path `$fpath`
- starts with a line like `"#compdef foo"`

list completion: code

`_rcc`

```
#compdef rcc
[[ -z $RCC_CMDS ]] && RCC_CMDS="$({...})"
cmds=($RCC_CMDS svquerclock ls2145dumps)
_arguments -S \
    '-N[specify a cluster ip]' \
    '-eg[show example for a specific command]' \
    '-f[add \-force to svctask commands]' \
    '-dc[use colon as output delimiter]' \
    '-d[show debug informat]' \
    '-nh[remove header from svcinfo output]' \
    '-h[show help message]' \
    '1:SVC/TBG command:${cmds})' && return 0
```

list completion: result

```
root@arcx325vf793# rcc -  
== option ==  
-N -- specify a cluster ip  
-d -- show debug informat  
-dc -- use colon as output delimiter  
-eg -- show example for a specific command  
-f -- add -force to svctask commands  
-h -- show help message  
-nh -- remove header from svcinfo output
```

```
root@arcx325vf793# rcc mk  
== SVC/TBG command ==  
mkarray          mkfcmap          mkpartnership    mksyslogserver   mkvdiskhostmap  
mkemailserver    mkhost          mkrcconsistgrp   mkuser              
mkemailuser      mkldapserver    mkrcrelationship mkusergrp           
mkfcconsistgrp   mkmdiskgrp      mksnmpserver     mkvdisk
```

descriptive completion: code

`_xcc`

```
#compdef xcc
cmds=(${(f) XCC_CMDS})
_arguments -C \
    '1:subcommand:->subcommand'
case $state in
    (subcommand)
        _describe -t xcc-commands 'XIV commands' cmds
        ;;
    esac
```

```
root@arcx325vf793# echo $XCC_CMDS T -n 2 ~ 6:28:10
vol_unformat: Unformats a volume by allocating a physical space on the disk drives for the content of t
his volume.
vol_unlock: Unlocks a volume, so that it is no longer read-only and can be written to.
root@arcx325vf793# ~ 6:28:22
```

descriptive completion: result

```
root@arcx325vf793# xcc vol_ ~ 6:28:22
== XIV commands ==
vol_by_id      -- Prints the volume name according to its specified SCSI serial number.
vol_copy       -- Copies a source volume onto a target volume.
vol_create     -- Creates a new volume.
vol_delete     -- Deletes a volume.
vol_delete_mirror_snapshots -- Deletes the last consistent and most updated mirroring snapshots of a
vol_disable_vaai -- Disables VAAI support for a specified volume
vol_enable_vaai -- Enables VAAI support for a specified volume
vol_format     -- Formats a volume.
vol_list       -- Lists all volumes or a specific one.
vol_lock       -- Locks a volume so that it is read-only.
vol_mapping_list -- Lists all hosts and clusters to which a volume is mapped.
vol_move       -- Moves a volume and all its snapshot from one Storage Pool to another.
vol_rename     -- Renames a volume.
vol_resize     -- Resizes a volume.
vol_set_wwn    -- Sets the WWN of a volume to be other than the system's WWN, used for
vol_unformat   -- Unformats a volume by allocating a physical space on the disk drives
vol_unlock     -- Unlocks a volume, so that it is no longer read-only and can be written
```


Summary

- Zsh is very suitable for interactive use
- Zsh has a huge amount of features
- Invest a small portion of time; you can get a very friendly shell environment
- Invest a huge amount of time; you can get every aspects of the shell be tuned

Any Questions?

Summary

- Zsh is very suitable for interactive use
- Zsh has a huge amount of features
- Invest a small portion of time; you can get a very friendly shell environment
- Invest a huge amount of time; you can get every aspects of the shell be tuned

Any Questions?