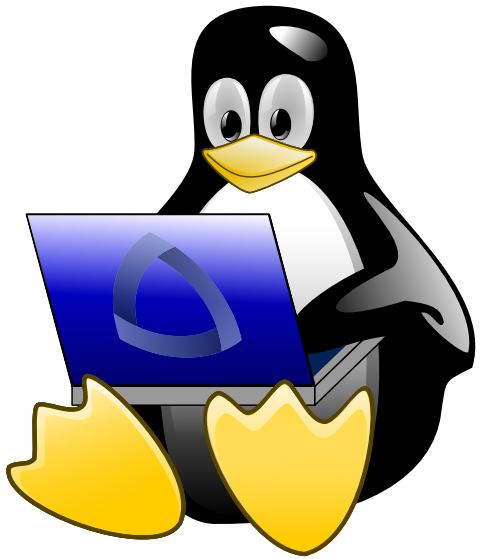
zsh Protips

Jack Rosenthal

4 April 2016



Colorado School of Mines
Linux Users Group

What is zsh?

zsh is a UNIX shell with bash-like syntax and plenty of features.

- zsh features its own line editor, zle, with bindable widgets and the ability to make custom bindings
- zsh features its own history expansion engine with Readline compatibility
- zsh tries to avoid bashisms by making the syntax do what it looks like it is doing (eg. appending and writing to two files in the same command)
- zsh comes with plenty of syntactic sugar and features like floating point arithmetic

What is zsh?

zsh is a UNIX shell with bash-like syntax and plenty of features.

- zsh features its own line editor, zle, with bindable widgets and the ability to make custom bindings
- zsh features its own history expansion engine with Readline compatibility
- zsh tries to avoid bashisms by making the syntax do what it looks like it is doing (eg. appending and writing to two files in the same command)
- zsh comes with plenty of syntactic sugar and features like floating point arithmetic

What is zsh?

zsh is a UNIX shell with bash-like syntax and plenty of features.

- zsh features its own line editor, zle, with bindable widgets and the ability to make custom bindings
- zsh features its own history expansion engine with Readline compatibility
- zsh tries to avoid bashisms by making the syntax do what it looks like it is doing (eg. appending and writing to two files in the same command)
- zsh comes with plenty of syntactic sugar and features like floating point arithmetic

What is zsh?

zsh is a UNIX shell with bash-like syntax and plenty of features.

- zsh features its own line editor, zle, with bindable widgets and the ability to make custom bindings
- zsh features its own history expansion engine with Readline compatibility
- zsh tries to avoid bashisms by making the syntax do what it looks like it is doing (eg. appending and writing to two files in the same command)
- zsh comes with plenty of syntactic sugar and features like floating point arithmetic

What is zsh?

zsh is a UNIX shell with bash-like syntax and plenty of features.

- zsh features its own line editor, zle, with bindable widgets and the ability to make custom bindings
- zsh features its own history expansion engine with Readline compatibility
- zsh tries to avoid bashisms by making the syntax do what it looks like it is doing (eg. appending and writing to two files in the same command)
- zsh comes with plenty of syntactic sugar and features like floating point arithmetic

Using zsh as your default shell (Personal machine)

```
$ chsh -s `which zsh`
```

Using `zsh` as your default shell (School machine)

Put this line at the top of your `.bash_profile`:

```
tty >/dev/null && command -v zsh >/dev/null && exec zsh
```

A few notes:

- You only have to do this because the `loginShell` LDAP attribute is used as your shell, and you don't have permission to change it.
- If you change it on one machine that uses `fermat` for its `/u`, then it will be changed on all.

Setting up zsh

Option 1

Use an "instantly awesome zsh" framework like *Oh My Zsh* or *Prezto*. This has the advantage of a *Vundle*-like plugin system and less time spent configuring.

Option 2

Do it yourself. Allows more customisation and typically lighter.

Writing a `.zshrc`

If you've decided to go with **Option 2**, you will need to write a `.zshrc`, a script that runs when you start `zsh`. Here are some things you may consider adding:

`bindkey -v` *or* `bindkey -e`	`vim` or `emacs` `zle` bindings
`HISTFILE=~/.histfile`	Persistent history
`HISTSIZE=1000`	Up to 1000 items in history
`SAVEHIST=1000`	Up to 1000 items persistent
`setopt appendhistory`	Append history to the history file
`setopt histignoredups`	Ignore duplicates in history
`setopt histignorespace`	Ignore lines which begin with a space
`setopt autopushd`	Use the dirstack as you `cd`

Also don't forget to set your `EDITOR`, `PAGER`, etc.

More `shopt` options

- `beep`/`nobeep` - Ring the terminal bell on `zle` error
- `notify` - Report the status of background jobs immediately
- `nomatch` - If a globbing pattern has no matches, print an error, instead of leaving it unchanged in the argument list.
- `autocd` - Change to a directory if you just type the name
- `correct` - Typo Correction
- `extendedglob` - Extended globbing, explained later

Read `man zshoptions`. There are many options.

Extended Globbing

With `setopt extendedglob`, you can use some cool extended globbing patterns:

- `**` matches any all of the child directories, recursively, including the current directory
- `***` is the same as above, but follows symlinks

If you enable `setopt globstarshort`, you can shorten `**/*` to `**` and `***/*` to `***`. This would cause `**.c` to match all files ending in `.c` recursively.

More Cool Globbing

Globbing What Its Not

```
zsh % ls
main.c      Makefile    README.md
zsh % echo ^*.c
Makefile README.md
```

Numeric Ranges

```
zsh % ls
hello1234   hello1235   hello1400
zsh % echo hello<1230-1240>
hello1234 hello1235
```

Perl-Style Or

```
zsh % ls
hello.txt   world.gif   zap.sh
zsh % echo *.(txt|gif)
hello.txt world.gif
```

More Cool Globbing

Globbing What Its Not

```
zsh % ls
main.c      Makefile    README.md
zsh % echo ^*.c
Makefile README.md
```

Numeric Ranges

```
zsh % ls
hello1234   hello1235   hello1400
zsh % echo hello<1230-1240>
hello1234 hello1235
```

Perl-Style Or

```
zsh % ls
hello.txt   world.gif   zap.sh
zsh % echo *.(txt|gif)
hello.txt world.gif
```

More Cool Globbing

Globbing What Its Not

```
zsh % ls
main.c      Makefile    README.md
zsh % echo ^*.c
Makefile README.md
```

Numeric Ranges

```
zsh % ls
hello1234   hello1235   hello1400
zsh % echo hello<1230-1240>
hello1234 hello1235
```

Perl-Style Or

```
zsh % ls
hello.txt   world.gif   zap.sh
zsh % echo *.(txt|gif)
hello.txt world.gif
```

Brace Expansion

```
zsh % echo hello_{one,two,three,four,five}
hello_one hello_two hello_three hello_four hello_five
zsh % echo hello{1..5}
hello1 hello2 hello3 hello4 hello5
zsh % echo hello{07..12}
hello07 hello08 hello09 hello10 hello11 hello12
zsh % echo {a..z}
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Brace Expansion

```
zsh % echo hello_{one,two,three,four,five}
hello_one hello_two hello_three hello_four hello_five
zsh % echo hello{1..5}
hello1 hello2 hello3 hello4 hello5
zsh % echo hello{07..12}
hello07 hello08 hello09 hello10 hello11 hello12
zsh % echo {a..z}
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Brace Expansion

```
zsh % echo hello_{one,two,three,four,five}
hello_one hello_two hello_three hello_four hello_five
zsh % echo hello{1..5}
hello1 hello2 hello3 hello4 hello5
zsh % echo hello{07..12}
hello07 hello08 hello09 hello10 hello11 hello12
zsh % echo {a..z}
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Brace Expansion

```
zsh % echo hello_{one,two,three,four,five}
hello_one hello_two hello_three hello_four hello_five
zsh % echo hello{1..5}
hello1 hello2 hello3 hello4 hello5
zsh % echo hello{07..12}
hello07 hello08 hello09 hello10 hello11 hello12
zsh % echo {a..z}
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Parameter Expansion

`${...}` is a parameter expansion. A parameter expansion will always involve a variable.

- `${VAR}` will expand to the value of `VAR`.
- `${#VAR}` will expand to the length of `VAR`.
- If `VAR` is a filename, `${VAR:h}` will expand to the directory of the file, `:t` to the name of the file, and `:r` to the file without its extension.
- `${VAR:s/`*find*`/`*replace*`/}` will do `sed`-style substitution

Read `man zshexpn` for more. I take no responsibility for brain damage.

Parameter Expansion

`${...}` is a parameter expansion. A parameter expansion will always involve a variable.

- `${VAR}` will expand to the value of `VAR`.
- `${#VAR}` will expand to the length of `VAR`.
- If `VAR` is a filename, `${VAR:h}` will expand to the directory of the file, `:t` to the name of the file, and `:r` to the file without its extension.
- `${VAR:s/`*find*`/`*replace*`/}` will do `sed`-style substitution

Read `man zshexpn` for more. I take no responsibility for brain damage.

Parameter Expansion

`${...}` is a parameter expansion. A parameter expansion will always involve a variable.

- `${VAR}` will expand to the value of `VAR`.
- `${#VAR}` will expand to the length of `VAR`.
- If `VAR` is a filename, `${VAR:h}` will expand to the directory of the file, `:t` to the name of the file, and `:r` to the file without its extension.
- `${VAR:s/`*find*`/`*replace*`/}` will do `sed`-style substitution

Read `man zshexpn` for more. I take no responsibility for brain damage.

Parameter Expansion

`${...}` is a parameter expansion. A parameter expansion will always involve a variable.

- `${VAR}` will expand to the value of `VAR`.
- `${#VAR}` will expand to the length of `VAR`.
- If `VAR` is a filename, `${VAR:h}` will expand to the directory of the file, `:t` to the name of the file, and `:r` to the file without its extension.
- `${VAR:s/`*find*`/`*replace*`/}` will do `sed`-style substitution

Read `man zshexpn` for more. I take no responsibility for brain damage.

Parameter Expansion

`${...}` is a parameter expansion. A parameter expansion will always involve a variable.

- `${VAR}` will expand to the value of `VAR`.
- `${#VAR}` will expand to the length of `VAR`.
- If `VAR` is a filename, `${VAR:h}` will expand to the directory of the file, `:t` to the name of the file, and `:r` to the file without its extension.
- `${VAR:s/`*find*`/`*replace*`/}` will do `sed`-style substitution

Read `man zshexpn` for more. I take no responsibility for brain damage.

Parameter Expansion

`${...}` is a parameter expansion. A parameter expansion will always involve a variable.

- `${VAR}` will expand to the value of `VAR`.
- `${#VAR}` will expand to the length of `VAR`.
- If `VAR` is a filename, `${VAR:h}` will expand to the directory of the file, `:t` to the name of the file, and `:r` to the file without its extension.
- `${VAR:s/`*find*`/`*replace*`/}` will do `sed`-style substitution

Read `man zshexpn` for more. I take no responsibility for brain damage.

Suffix Aliases

Say you want `.tex` files to open in your `$EDITOR`.

```
alias -s tex=$EDITOR
```

Then type just the filename as the command.

Global Aliases

Global aliases expand anywhere in the command.

```
alias -g ...='../..'
alias -g ....='../../..'
alias -g .....='../../../..'
```

Multiple Redirection

`zsh` can redirect to and from multiple inputs/outputs at the same time. So...

Rather than typing	You can type
`w >file1; w >file2; w >file3`	`w >file{1..3}`
`cat file{1,2} | less`	`less <file{1,2}`
`./server | tee log | grep ERR`	`./server >log | grep ERR`

Process Substitution

You can substitute a command in, like it's a file.

- `<(...)` is used to read output from a command
- `>(...)` is used to write to the stdin of a command
- `=(...)` is like `<(...)`, however creates a temporary file (so seek is allowed)

For example, to compare the output of two commands:

```
diff <(command1) <(command2)
```

Kill command for later use

Say you start typing a command but then realise you have to do something else first. Bind a key to `push-line`. I use `q` in `vi` normal mode:

```
bindkey -M vicmd q push-line
```

Then, press `<Esc>q` when you have another command to run first. Your old command will reappear when the first one finishes.

Making custom key widgets

Write a function, then bind it with `zle -N`. Example:

```
function __zkey_prepend_sudo {
    if [[ $BUFFER != "sudo "* ]]; then
        BUFFER="sudo $BUFFER"
        CURSOR+=5
    fi
}
zle -N prepend-sudo __zkey_prepend_sudo
bindkey -M vicmd "s" prepend-sudo
```

Now <Esc>s will put sudo at the beginning of the command.

Completion

To add intelligent completion to `zsh`, add this line to your `~/.zshrc`:

```
autoload -U compinit && compinit
```

Completion Automagic Rehash

```
zstyle ':completion:*' rehash true
```

Completion Case Correction

```
zstyle ':completion:*' matcher-list 'm:{a-z}={A-Z}'
```

Approximate Completion

Allow one error for every three letters typed.

```
zstyle ':completion:*:approximate:' max-errors
'reply=($(((($#PREFIX+$#SUFFIX)/3 )) numeric )'
```

Misc: Directory Hashing

Hash directories like their home direcories for quick and convenient access:

```
zsh % hash -d os=~/classes/cs/os
zsh % cd ~os
```

Protip

If you copy your `~/.zshrc` between systems, it may be convenient to set up hashes on a per system basis:

```
case $(hostname) in
    toilers)
        hash -d web=/home/www
        ;;
    mastergo)
        hash -d web=/var/www
        ;;
    ...
esac
```

Misc: Directory Hashing

Hash directories like their home direcories for quick and convenient access:

```
zsh % hash -d os=~/classes/cs/os
zsh % cd ~os
```

Protip

If you copy your `~/.zshrc` between systems, it may be convenient to set up hashes on a per system basis:

```
case $(hostname) in
    toilers)
        hash -d web=/home/www
        ;;
    mastergo)
        hash -d web=/var/www
        ;;
    ...
esac
```