Pimp your Shell

•••

CC BY-SA 2015 Nate Levesque

Why would you pimp your shell?

- Make it work the way you do!
  - Alias your typos, add commands, make it easier to look at
- Make it pretty
- Show more information
  - Add your Git status to your prompt when you’re in a Git repo
  - Show when you’re in an SSH session
  - Anything you can imagine!

How?

- Your shell loads a configuration file when you log in and open a shell
    - This file is a shell script
- If you can shell script, you can make your terminal do anything
    - Custom colors are super annoying!
- You can extend your shell using extensions such as Powerline, and depending on your shell with helpers like Oh-My-Zsh

The shell you use changes your pimping!

- Not all shells are compatible with each other
- Some shells are “drop-in” replacements for each other but have a number of extra options you may want
- Different shells may use different configuration files
  - bash uses ~/.bashrc, zsh uses ~/.zshrc
- Different systems may not have the same facilities (Mac OS, Cygwin, Linux, BSD, etc)

Let’s talk about some basics

Settings and Customizations

- In Linux-like systems, most of your personal configuration is stored in your home directory as “dotfiles”
- Some people choose to share their dotfiles on Github, and can be a good place to find cool things and inspiration
    - Typically these include settings for a variety of programs, including your shell

One Caveat to Shell Customizations

- Some applications use your shell non-interactively and customizations may break them in weird ways
    - CVS, SCP, and others
- Make sure to put this at the top of your shell config file so your customizations are only applied if a **user** is using your shell:

```
[[ $- != *i* ]] && return
```

Aliasing Commands

- Aliases are written as `<name of your alias>='the command'`
- You can alias any command including its command line flags

Some Useful Aliases

```
alias sl='ls -lr' # No damn steam locomotives here

alias ll='ls -l'

alias pgp='gpg'

alias pls='sudo !!' # this doesn’t work in all shells

alias KILLITWITHFIRE='kill -9'
```

Writing your own functions

- Your shell customizations are just a/some shell script/s!
- In shell scripts you can define your own functions, like with most programming languages
- Functions in your shell config are available by calling the name of the function in your shell
- Functions are defined (in Bash and Zsh) like:

  ```
  functionName(){

  }
  ```

- You can't do parameters as you would expect; you pass parameters by calling `functionName arg1 arg2` …
- You access passed parameters in your function using $1, $2, …

Some examples

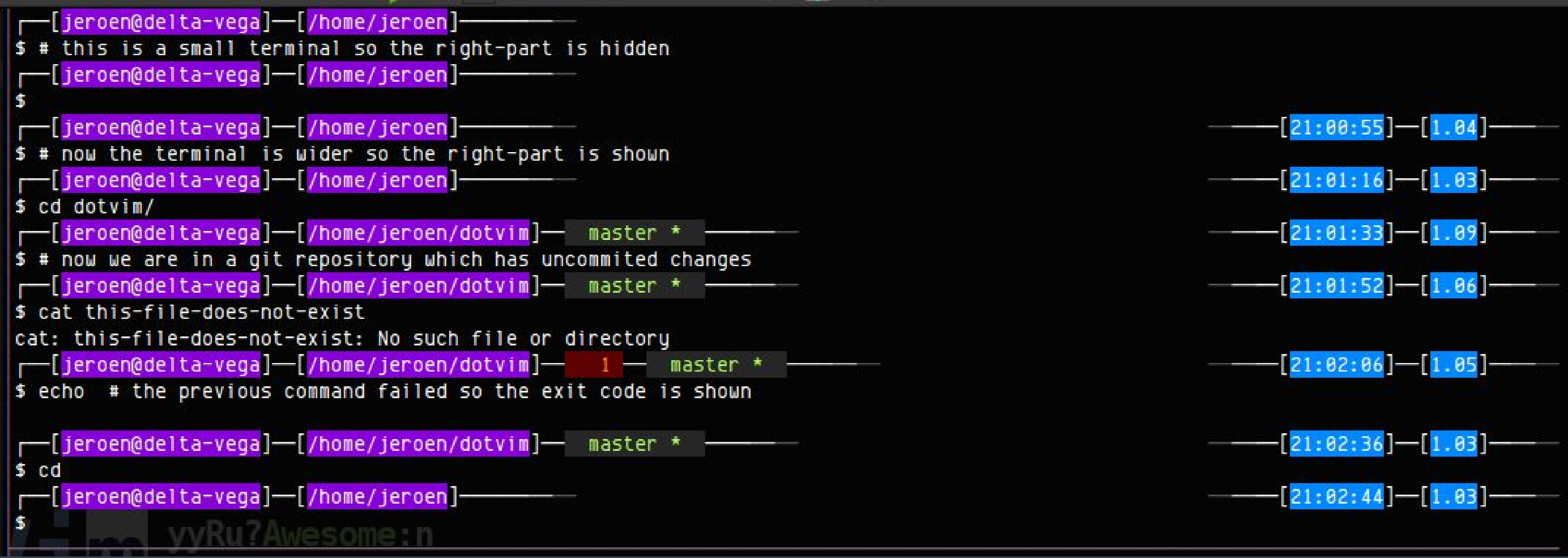
```
ufsshfs(){
    # Forcibly kills and unmounts an SSHFS
    # endpoint, caused by unfortunate
    # events where SSHFS is broken
    killall -KILL sshfs
    fusermount -u $1
}
```

```
mkGit(){
    # Sets up a git repository skeleton
    # with an initial README, LICENSE, and
    # .gitignore and performs an initial
    # commit after adding all the files.
    git init
    basename `pwd` > README.md
    echo -e "============" >> README.md
    echo -e '__pycache__' >> .gitignore
    echo -e 'bin' >> .gitignore
    echo -e 'LICENSE' >> LICENSE
    git add LICENSE README.md .gitignore
    git commit -m "Create repository
skeleton"
}
```

Customizing your Prompt

- Your prompt is stored in a variable called `$PS1` so you can change it just by setting the variable to another value
  - There is also PS2, PS3, and PS4 for other things, and are not as commonly customized
- You can add data to your prompt such as the time
- You can customize your prompt colors
  - This is annoying because these are defined using escape codes in some shells
  - You can also do gradients

A possibility of a custom Bash prompt



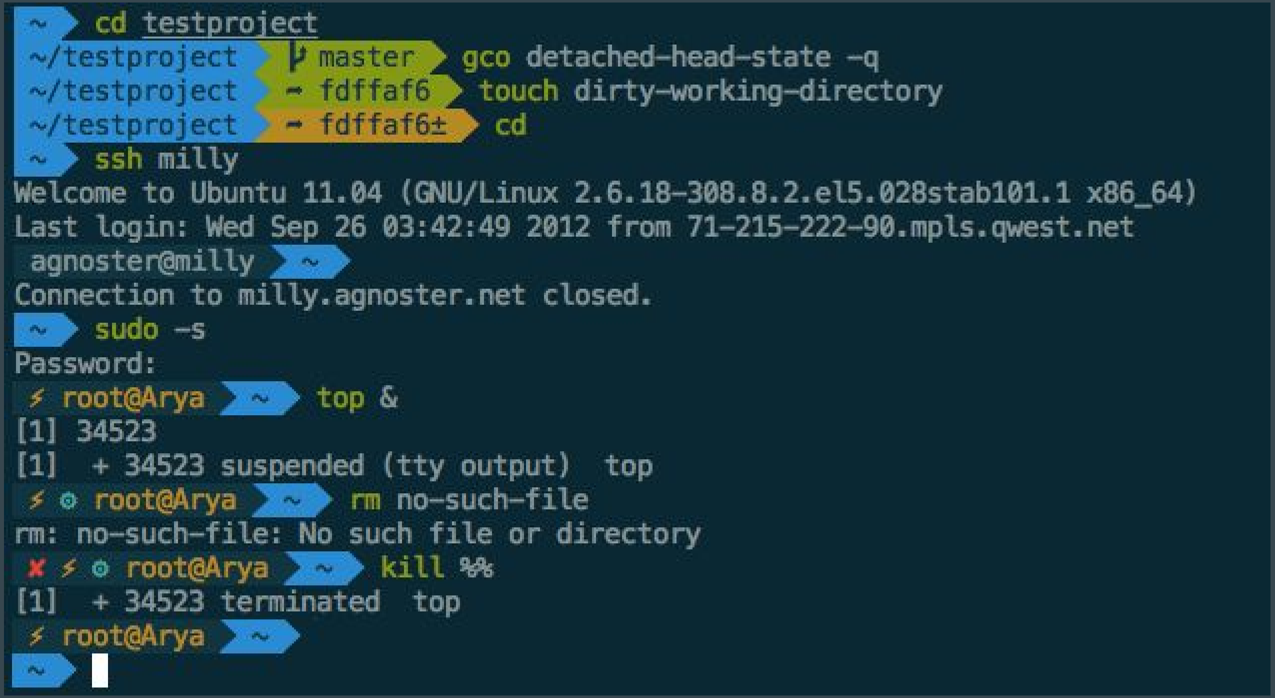
```
┌─[jeroen@delta-vega]─[/home/jeroen]────────
$ # this is a small terminal so the right-part is hidden
┌─[jeroen@delta-vega]─[/home/jeroen]────────
$

┌─[jeroen@delta-vega]─[/home/jeroen]────────                                                     ──────[21:00:55]─[1.04]─────
$ # now the terminal is wider so the right-part is shown
┌─[jeroen@delta-vega]─[/home/jeroen]────────                                                     ──────[21:01:16]─[1.03]─────
$ cd dotvim/
┌─[jeroen@delta-vega]─[/home/jeroen/dotvim]─  master * ────────                                  ──────[21:01:33]─[1.09]─────
$ # now we are in a git repository which has uncommited changes
┌─[jeroen@delta-vega]─[/home/jeroen/dotvim]─  master * ────────                                  ──────[21:01:52]─[1.06]─────
$ cat this-file-does-not-exist
cat: this-file-does-not-exist: No such file or directory
┌─[jeroen@delta-vega]─[/home/jeroen/dotvim]─  1 ─  master * ────────                             ──────[21:02:06]─[1.05]─────
$ echo  # the previous command failed so the exit code is shown

┌─[jeroen@delta-vega]─[/home/jeroen/dotvim]─  master * ────────                                  ──────[21:02:36]─[1.03]─────
$ cd
┌─[jeroen@delta-vega]─[/home/jeroen]────────                                                     ──────[21:02:44]─[1.03]─────
$
```

https://github.com/teranex/dotfiles/blob/master/bash/trexprompt

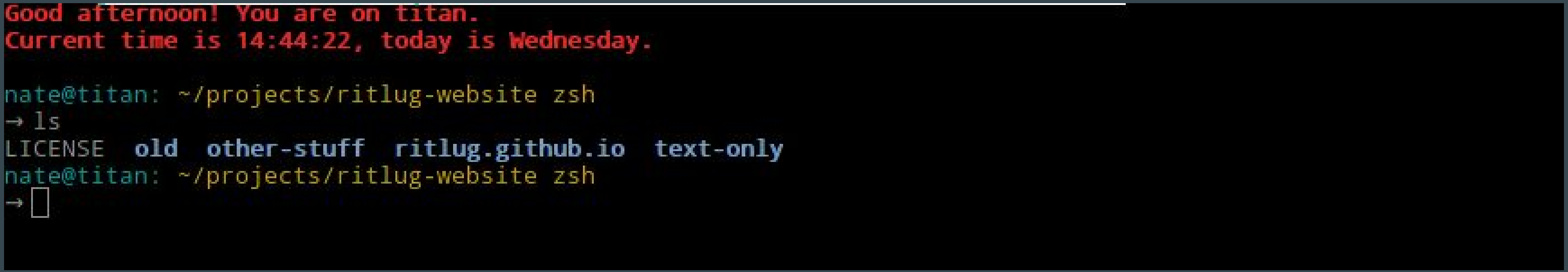
A possibility for a custom Zsh prompt



```
 ~  cd testproject
 ~/testproject  ⎇ master  gco detached-head-state -q
 ~/testproject  ➦ fdffaf6  touch dirty-working-directory
 ~/testproject  ➦ fdffaf6±  cd
 ~  ssh milly
Welcome to Ubuntu 11.04 (GNU/Linux 2.6.18-308.8.2.el5.028stab101.1 x86_64)
Last login: Wed Sep 26 03:42:49 2012 from 71-215-222-90.mpls.qwest.net
 agnoster@milly  ~ 
Connection to milly.agnoster.net closed.
 ~  sudo -s
Password:
 ⚡ root@Arya  ~  top &
[1] 34523
[1]  + 34523 suspended (tty output)  top
 ⚡ ⚙ root@Arya  ~  rm no-such-file
rm: no-such-file: No such file or directory
 ✘ ⚡ ⚙ root@Arya  ~  kill %%
[1]  + 34523 terminated  top
 ⚡ root@Arya  ~ 
 ~  █
```

https://gist.github.com/kevin-smets/8568070

Here’s mine!



```
Good afternoon! You are on titan.
Current time is 14:44:22, today is Wednesday.

nate@titan: ~/projects/ritlug-website zsh
→ ls
LICENSE  old  other-stuff  ritlug.github.io  text-only
nate@titan: ~/projects/ritlug-website zsh
→ 
```

https://github.com/thenaterhood/dotfiles/shellrc

You can add a custom greeting to your shell

- You may have noticed a welcome message on my shell
- This is separate from (and in addition to) your /etc/motd
- You can have your shell print anything when you log in

Cheatsheet

```
`readlink -f /proc/$$/exe` # Figure out what shell you’re running

if [ "$SSH_CONNECTION" != "" ]; then … # Do something if this is an SSH session

[[ $- != *i* ]] && return # Don’t apply customizations if not interactive
```

Things that seem like a good idea, but may not be

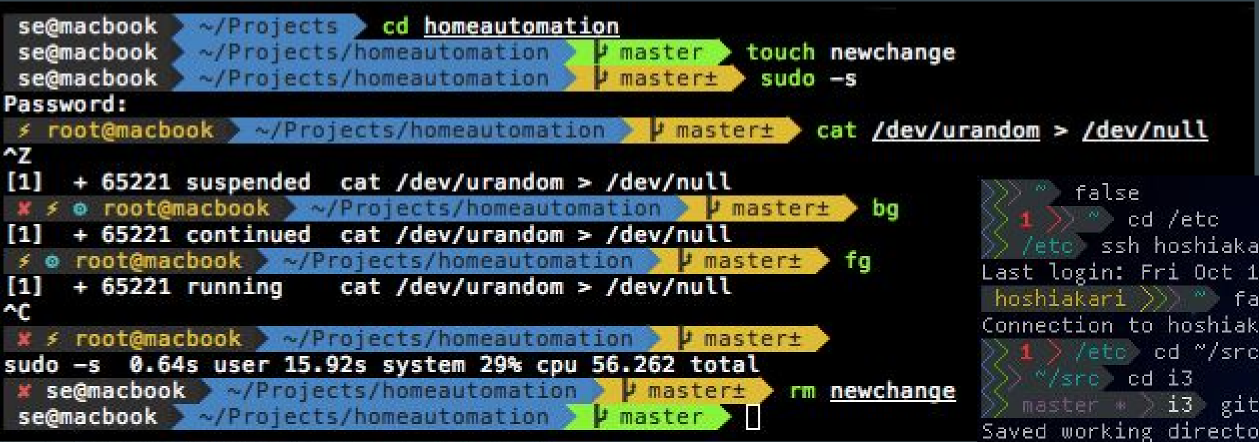
- Making your prompt super long - it may not fit on your screen
  - Putting your full current path in your prompt can be a problem, which is why my prompt is multiple lines
- Putting a lot of colors in your prompt
  - Colors don’t show up the same in all shells, and graphical shells may have a different background color
- Putting the time in your prompt - this *can* be useful, but remember that it’s the time your prompt printed, not necessarily the current time
- Removing the hostname from your prompt
  - This is how you accidentally reboot remote servers because you forgot what machine you’re on
- Using if statements to customize for multiple shells in one file, if you’re doing anything complex

Let’s talk about Powerline

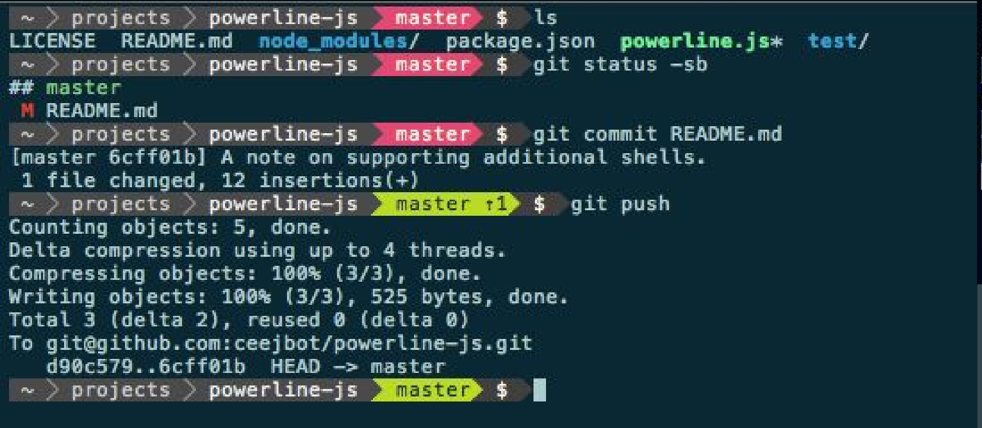
What is Powerline?

- Powerline is an extension (available for your shell, vim, tmux, and others) that gives you a quasi-graphical bar showing extra information
- Powerline is a popular way of customizing but can take some tweaking to get it to your liking
- You can get it at https://github.com/powerline/powerline and through pip (Powerline is written in Python)

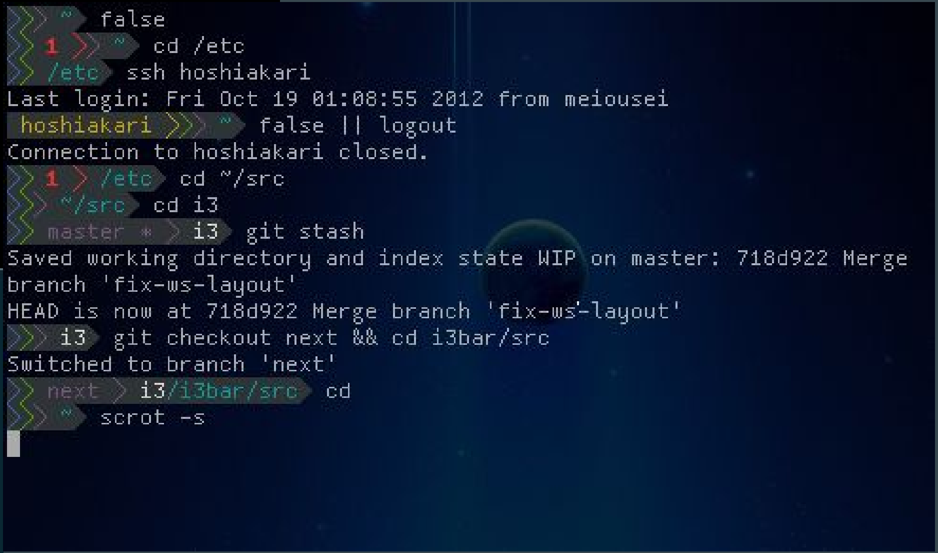
Powerline Prompts



```
se@macbook  ~/Projects  cd homeautomation
se@macbook  ~/Projects/homeautomation  ⭠ master  touch newchange
se@macbook  ~/Projects/homeautomation  ⭠ master±  sudo -s
Password:
 ⚡ root@macbook  ~/Projects/homeautomation  ⭠ master±  cat /dev/urandom > /dev/null
^Z
[1]  + 65221 suspended  cat /dev/urandom > /dev/null
 ✘ ⚡ ⚙ root@macbook  ~/Projects/homeautomation  ⭠ master±  bg
[1]  + 65221 continued  cat /dev/urandom > /dev/null
 ⚡ ⚙ root@macbook  ~/Projects/homeautomation  ⭠ master±  fg
[1]  + 65221 running    cat /dev/urandom > /dev/null
^C
 ✘ ⚡ root@macbook  ~/Projects/homeautomation  ⭠ master±
sudo -s  0.64s user 15.92s system 29% cpu 56.262 total
 ✘ se@macbook  ~/Projects/homeautomation  ⭠ master±  rm newchange
se@macbook  ~/Projects/homeautomation  ⭠ master  
```



```
 ~ > projects > powerline-js  master  $ ls
LICENSE  README.md  node_modules/  package.json  powerline.js*  test/
 ~ > projects > powerline-js  master  $ git status -sb
## master
 M README.md
 ~ > projects > powerline-js  master  $ git commit README.md
[master 6cff01b] A note on supporting additional shells.
 1 file changed, 12 insertions(+)
 ~ > projects > powerline-js  master ↑1  $ git push
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 525 bytes, done.
Total 3 (delta 2), reused 0 (delta 0)
To git@github.com:ceejbot/powerline-js.git
   d90c579..6cff01b  HEAD -> master
 ~ > projects > powerline-js  master  $ 
```



```
    ~  false
  1    ~  cd /etc
  /etc  ssh hoshiakari
Last login: Fri Oct 19 01:08:55 2012 from meiousei
 hoshiakari    ~  false || logout
Connection to hoshiakari closed.
  1  /etc  cd ~/src
   ~/src  cd i3
  master *  i3  git stash
Saved working directory and index state WIP on master: 718d922 Merge
branch 'fix-ws-layout'
HEAD is now at 718d922 Merge branch 'fix-ws-layout'
    i3  git checkout next && cd i3bar/src
Switched to branch 'next'
  next  i3/i3bar/src  cd
    ~  scrot -s
```

Installing Powerline

- Powerline requires Python, GCC, and other packages depending on the setup you want
- Powerline is available in pip as powerline-status and in some package managers as python-powerline
- After installing the package manually, run

  ```
  ln -s {path_to_powerline}/scripts/powerline ~/.local/bin
  ```

- On Arch, Powerline is available in the AUR and will do all the necessary setup for you and install itself for Vim automatically (though this may not work correctly)

Installing Powerline's fonts (if you installed manually)

- Powerline uses patched fonts (glyphs) which need to be configured

```
wget https://github.com/powerline/powerline/raw/develop/font/PowerlineSymbols.otf

wget https://github.com/powerline/powerline/raw/develop/font/10-powerline-symbols.conf

mv PowerlineSymbols.otf ~/.fonts/

fc-cache -vf ~/.fonts/

mv 10-powerline-symbols.conf ~/.config/fontconfig/conf.d/
```

Using the defaults (On Arch with Zsh)

Run `. /usr/share/zsh/site-contrib/powerline.zsh`

- This will start Powerline in your shell for the current session only
- You can use powerline all the time by adding that line to your shell configuration
- On other distros, the path may vary depending how and where you installed
- You may need to upgrade (g)vim to (g)vim-python3 in order to fix an error before using the Python3 version of Powerline in Vim

Powerline Configuration Files

- Main configuration
  - `powerline/config.json`
- Colorschemes
  - `powerline/colorschemes/{name}.json`
  - `powerline/colorschemes/{extension}/__main__.json`
  - `powerline/colorschemes/{extension}/{name}.json`
- Themes
  - `powerline/themes/top_theme.json`
  - `powerline/themes/{extension}/__main__.json`
  - `powerline/themes/{extension}/default.json`

Powerline documentation is at powerline.readthedocs.org

Let’s talk about Oh-My-Zsh

What is Oh-My-Zsh?

- An open source framework for managing your Zsh configuration
- Has 200+ extensions
- Has 140+ themes
- Auto-updates!

Installing Oh-My-Zsh

```
sh -c "$(wget https://raw.github.com/robbyrussell/oh-my-zsh/master/tools/install.sh -O -)"
```

- Note: I do NOT advocate for running scripts directly from the Internet as this command does. Make sure you know what you're running
- Oh-My-Zsh may also be available in your package manager (it's available for Arch in the AUR)
- Oh-My-Zsh is for Zsh and does not work with other shells (but can be used for inspiration!)

Configuring Oh-My-Zsh

- Edit your ~/.zshrc file
- Specify plugins by adding `` `plugins=({your}, {plugins})` `` or by adding to the array
- Set a theme by setting `ZSH_THEME` to the theme of your choice
  - According to the documentation, “if you’re feeling feisty”, you can also set this to “random”

Oh-My-Zsh documentation is at github.
com/robbyrussell/oh-my-zsh

Closing Thoughts

- If you have a lot of dotfiles, writing an installer script isn’t necessarily a bad idea
    - I have one so that I can clone my dotfiles on a new system, run ./install.sh and be good to go
- Your distribution may ship with certain settings out of the box, either in your home directory or in /etc
- You can configure most things systemwide by putting their settings in /etc
- There are many ways to do things. What works for someone else may not work for you
- Make sure you still have a general idea of how to do things without your customizations, since sometimes you can’t use them
- Follow conventions for your dotfiles! It makes it easier to push them out if you work at a company that allows you to push them via their automation

Questions, Comments, Concerns...