

# WHAT'S IN MY RC 2021

5/29/2021 - Henry Unite

Welcome to a tour of my `.zshrc` file! I thought it would be fun to take a dive into the different tools I use to elevate my developer game.

**If you want to see my whole config, I keep all my setup files on GitHub:**

<https://github.com/unitehenry/config>

## TOOLS I USE

Before we dive into my configuration, here are the tools I use that help me speed up my tasks as a developer:

`fzf` | [Fuzzy File Finder](#)

`pandoc` | [Pandoc Document Converter](#)

`brew` | [MacOS Package Manager](#)

## THE VARIABLES

```
# EDITOR
export EDITOR="vi";
export VISUAL="vi";

# iCloud Directory
export DOCS="/Users/henryunite/Library/Mobile
            Documents/com~apple~CloudDocs";

# Work Directory
export WORK="/Users/henryunite/Projects/bicycletransit";
```

Pretty straightforward, but these are what I use to:

- Default editing to be opened in vim
- Reference my iCloud directory which I use to keep all my personal files
- A quick reference to where I keep all my work repositories, notes, projects

## CREDENTIALS

```
# Credentials Fetcher
function username() {
    export PASS_BACK_PATH=$(pwd);
    cd $DOCS/passwords;
    echo $(decrypt-file $(fzf) | grep "Username:" | cut -d ":" -f2) | pbcopy;
    cd $PASS_BACK_PATH && unset PASS_BACK_PATH;
}

function password() {
    export PASS_BACK_PATH=$(pwd);
    cd $DOCS/passwords;
    echo $(decrypt-file $(fzf) | grep "Password:" | cut -d ":" -f2) | pbcopy;
    cd $PASS_BACK_PATH && unset PASS_BACK_PATH;
}
```

There are so many chrome extensions, keychains, any ways to access your passwords. I personally encrypt my passwords in my cloud storage so I can access them by utilizing a `aes-256-cbc` decryption tool.

## FILE FORMATTING

```
## Code Formatter
function format-file() {
    export FILENAME="$(basename $@)";
    export EXTENSION="${FILENAME##*.}";

    if [ $EXTENSION = 'py' ]
    then
        yapf --in-place $@;
        return 0;
    fi

    if [ $EXTENSION = 'php' ]
    then
        php-cs-fixer fix $@;
        rm .php_cs.cache;
        return 0;
    fi

    npx prettier --write --single-quote $@;

    unset FILENAME; unset EXTENSION;
}
```

File formatter that handles the languages I use on a day-to-day basis. It gets the job done for most file types including JSON, YAML, and even markdown.

## SPELL CHECK

```
## Spellcheck
function spellcheck-file() {
  npx spellchecker-cli --files $@;
}
```

When you're writing as much markdown documentation as me, you'll want an easy way to spell check your files.

## WHAT THE COMMIT

```
## What the Commit
function wtf() { git commit -am "$(curl
  http://whatthecommit.com/index.txt)"; }
```

This is a gimmick, but if you ever just want to commit file changes and you just don't know what to say in the commit message, [what the commit](http://whatthecommit.com/index.txt) is just a fun resource to get whacky commit messages.

## CHEAT SHEET

```
## Cheat
function cheat(){ curl https://cheat.sh/"$@"; }
```

There are so many times I use a CLI tool and can't remember simple commands and options that it takes to perform certain tasks. Check out [cheat.sh](https://cheat.sh) if you're looking for an easy way to reference different CLI tools.

## DOCUMENT GENERATION

```
## Generate Markdown
function generate-doc() {
  cp -rf . /tmp;
  if [ -n "$2" ]
  then
    pandoc -s $1 -c $2 -o "/tmp/$1.html";
  else
    pandoc -s $1 -o "/tmp/$1.html";
  fi
  open "/tmp/$1.html";
}

## Generate Slide
function generate-slide() {
  # https://revealjs.com/config/
  pandoc -t revealjs \
```

```
-V progress="false" \  
-V navigationMode="linear" \  
-V transition="none" \  
-s $1 -o "/tmp/$1.html";  
cp -rf . /tmp;  
open "/tmp/$1.html";  
}
```

I use markdown to write documentation all the time, but if I need to send a coworker a document or present a slide with content that is written in markdown, I'll use pandoc to generate these intermediary file formats.

It's really nice to leverage CSS when I want to make my documents look nice or need a clean way to look at markdown files.

## FILE ENCRYPTION

```
## Encrypt : aes-256-cbc  
function encrypt-file() {  
    if [ -z $@ ]  
    then  
        echo -n "Enter Encrypt Phrase: "; read -s ENCRYPTINPUT; echo "\n";  
        echo $ENCRYPTINPUT | openssl enc -aes-256-cbc;  
        unset ENCRYPTINPUT;  
    else  
        openssl enc -aes-256-cbc -in $@;  
    fi  
}  
  
## Decrypt : aes-256-cbc  
function decrypt-file() {  
    if [ -z $@ ]  
    then  
        openssl enc -d -aes-256-cbc;  
    else  
        openssl enc -aes-256-cbc -d -in $@;  
    fi  
}
```

It's nice to have a quick way to encrypt and decrypt files with sensitive information.

## HOME BREW

```
## Homebrew Install Script  
function install-homebrew() { /bin/bash -c "$(curl -fsSL  
    https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)";  
}
```

This is just in my configuration in the event that I just want to install homebrew without copying and pasting the install script from the website.

If I have a new Mac I need to setup, it'll make the setup so much easier.

## VERSION MANAGERS

```
source ~/.nvmrc;  
source ~/.rvmrc;
```

I've been using `nvm` and `rvm` to manage my node and ruby installations. They append rc scripts to load into your base rc file which I extract into their own designated files and load them in at the end.