What's The Point?

If you aren't convinced it's worth your time to put your dotfiles into Git version control, consider this:

By storing your dotfiles in a Git repository, you'll be able to use them on any OS X or Linux machine with Internet access.

This means that in addition to gaining the ability to revert back to a known-working setup should you misconfigure your files, you will also be able to work in an environment you've customized yourself. On almost any workstation or server, you're a simple `git clone` away from the familiarity of your customizations. More on `git clone` later… For now, we'll begin with an example.

A Basic .vimrc

The following is an example of the type of file we would manage with `git`. This is actually an abridged version of my own `.vimrc`. The full version is available for view on my public Github:

```
set nocompatible         " get rid of Vi compatibility mode. SET FIRST!
filetype plugin indent on " filetype detection[ON] plugin[ON] indent[ON]
set t_Co=256             " enable 256-color mode.
syntax enable            " enable syntax highlighting (previously syntax on).
colorscheme desert       " set colorscheme
set number               " show line numbers
set laststatus=2         " last window always has a statusline
filetype indent on       " activates indenting for files
set nohlsearch           " Don't continue to highlight searched phrases.
set incsearch            " But do highlight as you type your search.
set ignorecase           " Make searches case-insensitive.
set ruler                " Always show info along bottom.
set autoindent           " auto-indent
set tabstop=4            " tab spacing
set softtabstop=4        " unify
set shiftwidth=4         " indent/outdent by 4 columns
set shiftround           " always indent/outdent to the nearest tabstop
set expandtab            " use spaces instead of tabs
set smarttab             " use tabs at the start of a line, spaces elsewhere
set nowrap               " don't wrap text
```

If you don't have your own `.vimrc` in your home directory, you're welcome to use mine. It is

intentionally minimal, as it works standalone (you don't need to do anything other than drop it into your home directory for it to work). That said, we're going to do things a little differently from here on, so pay attention (if you aren't already).

Organization, Organization, Organization

The typical location for dotfiles on an OS X or Linux machine would be in a users home directory, e.g. `/home/smalleycreative/.vimrc`. We aren't typical though, are we? We are trying to be crafty.

For starters, we'll be putting all of our dotfiles into a folder called `dotfiles`, like so: `/home/smalleycreative/dotfiles/vimrc`. Then, we'll simply symlink to them from our home directory. To programs like `vim` and `bash` these symlinks are transparent. As far as these programs are concerned, our dotfiles will still appear to exist at the top-level of our home directory, even though they'll be tucked away in the `dotfiles` directory.

If you're on your OS X or Linux workstation now, you can get started by creating the `dotfiles` directory in your home directory by running the following command:

```
mkdir ~/dotfiles
```

Place your `.vimrc` within `~/dotfiles` (and/or any other dotfiles you want to manage with Git). For simplicity, I drop the dot from the filename (`.vimrc` becomes `vimrc`). I only prepend the dot when I create my symlink to `vimrc` in my home directory. What is important to keep in mind is we still haven't used Git at all yet. We're just preparing by organizing our dotfiles into one folder. For example, to copy an existing `.vimrc`, run the following command:

```
mv ~/.vimrc ~/dotfiles/vimrc
```

Once we have a bunch of dotfiles in this directory, a directory listing will likely look a lot like this:

```
ls ~/dotfiles
vimrc
zshrc
bashrc
```

The Install Script

Since we want portability (the ability to use our files on any machine with Internet access), we're going to need an install script. This script goes in our `~/dotfiles` directory. Here's mine (The full version is available for view on my public Github):

```bash
#!/bin/bash
############################
# .make.sh
# This script creates symlinks from the home directory to any desired dotfiles in
~/dotfiles
############################

########## Variables

dir=~/dotfiles                      # dotfiles directory
olddir=~/dotfiles_old               # old dotfiles backup directory
files="bashrc vimrc vim zshrc oh-my-zsh"    # list of files/folders to symlink in
homedir

##########

# create dotfiles_old in homedir
echo "Creating $olddir for backup of any existing dotfiles in ~"
mkdir -p $olddir
echo "...done"

# change to the dotfiles directory
echo "Changing to the $dir directory"
cd $dir
echo "...done"

# move any existing dotfiles in homedir to dotfiles_old directory, then create sym
links
for file in $files; do
    echo "Moving any existing dotfiles from ~ to $olddir"
    mv ~/.$file ~/dotfiles_old/
    echo "Creating symlink to $file in home directory."
    ln -s $dir/$file ~/.$file
done
```

The commented sections in the above script explain it. First, the script cleans up any old symlinks that may exist in our home directory and puts them into a folder called `dotfiles_old`. It then iterates through any files in our `~/dotfiles` directory and it creates symlinks from our home directory to these files. It handles naming these symlinks and prepending a dot to their filename.

If we've got this script in our dotfiles directory, and we've got our dotfiles in there too, we're ready to start using Git to manage these files.

Gitting Started with GitHub (Git it?)

Before continuing, we're going to want to create a GitHub account. GitHub allows us to set up a free public Git repository, accessible from any machine with Internet access. They also have a very well-written help section. You would be well advised to read through and follow the instructions pertaining to your particular operating system:

- GitHub Beginner's Guide for OS X
- GitHub Beginner's Guide for Linux

Seriously — Read or at least skim over this documentation. Once you've set up your account, and you're comfortable with the basics, come back here, and we'll continue with the steps required to get your dotfiles stored in your public Git repository at GitHub.com.

Creating Our Local Git Repo

On our workstation we are going to initialize a new Git repo. To make our `~/dotfiles` directory a Git repo, we simply change to it, and run the `git init` command:

```
cd ~/dotfiles
git init
```

We then start to track any files that we want to be a part of our repo by using the `git add` command on them:

```
git add makesymlinks.sh
git add vimrc
```

Finally, once we're happy with the state of our files, we can commit them.

> **Think of a commit as a snapshot of your project —code, files, everything — at a particular point in time. More accurately, after your first commit, each subsequent commit is only a**

snapshot of your changes. For code files, this means it only takes a snapshot of the lines of code that have changed. For everything else like music or image files, it saves a new copy of the file.

```
git commit —m 'My first Git commit of my dotfiles'
```

What's important to realize is that this commit is local — We still haven't uploaded *anything* to GitHub. In fact, we cannot do this until we tell our local repository *where* our public GitHub repository actually resides. To do this, we use `git remote add origin`, like so:

```
git remote add origin git@github.com:mygithubusername/dotfiles.git
```

Finally, we can **push** our changes to GitHub:

```
git push origin master
```

Tracking Updates to Our Git Repo

If we decide to update our `vimrc`, we're going to want to make sure these changes get tracked on GitHub.com, so what is the process? First, add the file:

```
git add vimrc
```

Then, commit locally, and write a commit message:

```
git commit —m 'Changed vim colorscheme!'
```

Then, push to GitHub:

```
git push origin master
```

Cloning Our Dotfiles to Another Machine

Ultimately, we're going to come to a point where we want to use our customized dotfiles on

another server or workstation. That is, after all, one of the primary benefits of using Git to manage our dotfiles. To do this, it's as simple as running the following command from our home directory:

```
git clone git://github.com/<mygithubusername>/dotfiles.git
```

Once the repository has been cloned to our home directory, simply change to the `dotfiles` directory, make the `makesymlinks.sh` script executable, and run the script, like so:

```
cd ~/dotfiles
chmod +x makesymlinks.sh
./makesymlinks.sh
```

Warning: If you're running Debian Linux, this will most likely install `zsh` on your system if it isn't already installed. This is a feature I added to my script to automate things as much as possible. If for some reason you don't want `zsh` to be installed on your computer (e.g. you're on somebody else's server), remove the `install_zsh` line from `makesymlinks.sh`. This will cause it to set up all of your dotfiles without attempting to install `zsh`.

That's it! If the settings don't take effect right away, we can just log out and log back in (this will re-source our dotfiles).

Updating a Local Git Repo

The best way to explain this is with a scenario. Say we've created a repository on our workstation (Machine A) and we've pushed our changes to GitHub, then we've cloned these changes down to our server (Machine B). We then go back onto our workstation (Machine A), make more changes, and push them to GitHub. How do we get these changes onto our server (Machine B)? To do this, we use the `git pull` command:

```
git pull origin master
```

After running this command, Machine B will be current with any changes that were pushed to GitHub from Machine A. It quite literally **pulls** any updates to the repo from GitHub.

Colophon

Whether you're trying to wrangle a large collection of custom dotfiles or you're just getting started and you weren't sure where to begin, I hope this was an informative introduction to managing your configuration settings via the power of Git.