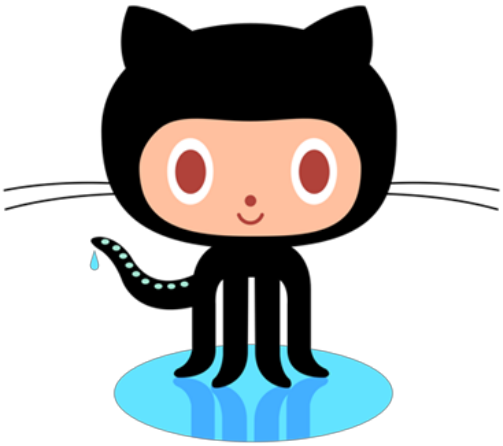
Fun with ones and zeros!

Using Git and Github to Manage Your Dotfiles

by Michael Smalley on January 12, 2012 in Tutorials with 36 Comments



If you find this post useful, please consider donating in the form of a

If you use OS X or Linux on your desktop/servers, you may be at a point where you have configured a lot of your own settings, configurations, or themes within dotfiles. For the uninformed, dotfiles are files in your home directory that begin with a dot, or full-stop character. This indicates to the operating system that they are hidden files, used to set configuration settings for tools like `vim`, or shells such as `bash` and `zsh` to name a few.

This tutorial does not go into the specifics of configuring your dotfiles. Instead, my goal is to provide you with a light introduction to Git version control, allowing you to maintain your dotfiles in a centralized repository on github.com

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):

```
#!/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 symlinks
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.

Share this with:

Michael Smalley

Michael is the creator and main author of the Smalley Creative Blog. He is a guy who loves technology (particularly open source), educating people about technology, and working with people who love technology as much as he does. Follow him on Twitter @michaeljsmalley.

Tagged: configuration, dotfile, dotfiles, git, github, management

36 Comments on "Using Git and Github to Manage Your Dotfiles"

1.  *amar* says:

   February 24, 2012 at 6:14 AM

Totally awesome article.

Thanks
Keep the good work.

Reply

-  *Michael Smalley* says:
  February 26, 2012 at 1:42 PM

  I'm glad you enjoyed it. Thank you for the kind words!

  Reply

2.  *Saiyasodharan* says:
   March 26, 2012 at 7:28 AM

   I have been saving all my config files in gmail drafts since now and its been a mess. Thanks to you for this article. I now moved all those files to github.

   Reply

   -  *Michael Smalley* says:
     March 26, 2012 at 7:34 AM

     I'm glad you found it useful. You should now find managing your configurations across multiple machines a lot more convenient and quick!

     Reply

3.  *lwm* says:
   April 23, 2012 at 1:57 PM

   Amazing stuff!
   Thanks for getting me off the ground on this one!

   Reply

-  *Michael Smalley* says:
  April 27, 2012 at 6:21 PM

  I'm glad you found this informative and useful. Thanks for reading!

  Reply

4.  *nathaniel* says:
   April 24, 2012 at 1:52 PM

   Good article & well written. Shame you neglected to mention anywhere in the article that if someone used your install script it would install zsh at the same time. I guess it's a good thing I read the code.

   Reply

   -  *Michael Smalley* says:
     April 27, 2012 at 6:20 PM

     I'm glad you enjoyed the article. You'll be pleased to know that I've added a warning for anyone who would dare run any script from the Internet without reading the actual code contained within. To anyone reading this, running scripts without taking a glance at them to see what they do does you a disservice on two fronts. Firstly, you're skipping an opportunity to learn from somebody else's code. Secondly, you're putting your system/network in jeopardy by running something on it that you don't fully understand. The point is, take a note from Nathaniel. If you download a script from the Internet, read it before you run it!

     Reply

5.  *Todd Partridge (Gen2ly)* says:
   May 26, 2012 at 8:32 AM

   Want to add too that this is a great article. Git's always been one of those 100 pound gorillas to me, so very much appreciate it.

   Reply

-  *Michael Smalley* says:
  June 3, 2012 at 5:13 PM

  Git is challenging for everyone who wants to do more than simple clone operations. I'm glad this helped!

  Reply

6.  *Granze* says:
   September 19, 2012 at 8:36 AM

   Thanks for sharing. Really interesting stuff!

   Reply

   -  *Michael Smalley* says:
     October 11, 2012 at 10:13 AM

     I'm glad you found it useful.

     Reply

7.  *Pedram* says:
   November 3, 2012 at 8:41 AM

   Thanks so much for publishing this! I've been telling myself I will get my dotfiles onto git forever and now I finally have.

   Reply

   -  *Michael Smalley* says:
     November 4, 2012 at 6:13 PM

     I'm glad to hear I've helped you make the jump!

     Reply

8.  *jm* says:
November 28, 2012 at 8:03 PM

Solid. For a while I'd been thinking of doing the same thing, and you've gone and done it!

Currently my home directory itself is a git repo where everything is ignored and files have to be force-added to track them. Having the root of one's home tree be a git repository is problematic, however.

Reply

9. 

Patrick says:
January 2, 2013 at 1:02 PM

Thank you so much. This post was phenomenal, I sat down today thinking "Man, how am I going to get my .vimrc into gitHub.. maybe symlinks?".
Then on my first goole your post comes up and blows me away. Thank you so much for taking the time to make this script and to share it with such a well laid out post.

Reply

-  *Michael Smalley* says:
January 2, 2013 at 2:06 PM

Comments like yours are the reason I do what I do here. Thank you!

Reply

10.  *Ara* says:
March 4, 2013 at 3:19 AM

Great article! Exactly what I was looking for, thanks!

Reply

-  *Michael Smalley* says:

March 5, 2013 at 10:45 AM

I'm glad it came in handy for you!

Reply

11.  *Marcus* says:
March 18, 2013 at 12:16 PM

Thanks for this great and short introduction!!
This was exactly what I've been looking for.

Reply

-  *Michael Smalley* says:
March 20, 2013 at 9:43 PM

You're quite welcome. I'm glad I was able to help!

Reply

12.  *Hans* says:
March 20, 2013 at 8:02 PM

Thx for this great article. I saw a lot people sharing their dotfiles at github and I always was thinking how they do it. Because they won't use a large .gitignore file to exclude all their stuff in ~/

Your article really helps and later I will have a look at your files 

But one question: I use the bash an my mac und like it. Why do you prefer zsh?

Reply

-  *Michael Smalley* says:
March 20, 2013 at 9:42 PM

I prefer zsh because I use `oh-my-zsh`, a framework that adds a lot of functionality to the standard `bash` you may be used to. Give it a look here, I'm sure you'll be convinced after reading the feature-set:

https://github.com/robbyrussell/oh-my-zsh

Reply

-  *Hans* says:
  March 20, 2013 at 11:37 PM

  Oh-my-zsh has some nice features. I'll give it a try but I'm not fully convinced. Maybe I'm not this hardcore user and I don't see the real advantages 😉

  Reply

13.  *Abdullah Al Mamun* says:
May 24, 2013 at 6:32 AM

Thanks a lot for sharing. Just pushed my dotfiles to github repo. 😀

Reply

14.  *David Weinraub* says:
July 9, 2013 at 3:45 AM

Another "Me, too" post thanking you for this excellent tutorial.

For a long while now, I've been wanting to push my dotfiles down into a single folder and get them under source control, but somehow just mentally missed that symlinking them up into the home directory would keep everything working fine. Your automated symlink script is the cherry on top.

Thanks again! 😉

Reply

-  *Michael Smalley* says:
  July 16, 2013 at 8:32 PM

  Thank you for the kind words! I'm glad I was able to help!

  Reply

15.  *Vidit* says:
[July 17, 2013 at 2:09 PM]

I'm setting up my own dotfile git repo, and this post was a great help.

I have a small question though, does it matter whether the file and directory names in the repo have '.' in them? I have seen many github dotfiles repo which have them, but all the blogs suggest otherwise.

I understand the 'make_symlinks' scripts will need modification, but I'm not worried about that. I'm confused about the naming.

Reply

-  *Michael Smalley* says:
[July 25, 2013 at 10:32 AM]

It does matter. As long as the symlinks begin with a dot, the files they point to can be called anything. You'll notice this is how my implementation works. For example, I have the `~/.bashrc` symlink pointing to a file called `~/dotfiles/bashrc`. Technically, `~/dotfiles/bashrc` could be called anything — as long as when the symlink gets created (in the setup script) it points to the correct file within the `~/dotfiles` directory.

Thank you for your question!

Reply

16.  *Richard "RichiH" Hartmann* says:
[July 25, 2013 at 8:27 AM]

Hi Michael,

this post has been referenced in a discussion about managing dotfiles with Git in freenode's #git channel just now. I have to say it's quite a concise and to the point howto; kudos for that.

I was wondering if you ever tried vcsh[1]? This would allow you to ditch the dotfiles directory and maintain your dotfiles directly in $HOME. No need for symlinks or other hacks; just clean Git repositories happily sharing $HOME and not interfering with each other.

Even if you don't get around to try it or if you don't like it I'd like to thank you for pushing the use of Git for non-code purposes. It's a bit of a passion of mine and it's nice to see that others Care as well.

Richard

[1] https://github.com/RichiH/vcsh

Reply

-  *Michael Smalley* says:
  July 25, 2013 at 10:43 AM

  Richard,

  Thank you for the compliment. This is the first I've ever heard of vcsh. I'll definitely give it a fair shot at some point this week. It seems quite interesting!

  If you have a thing for using Git for non-code purposes, you'll probably enjoy this book (which I reviewed): http://www.amazon.com/Git-Version-everyone-Ravishankar-Somasundaram/dp/1849517525

  Reply

17.  *Michael Smalley* says:
    June 3, 2012 at 5:18 PM

    One of the biggest rewards I get from effective knowledge-sharing is seeing people build on what they've learned. For anyone interested, this is a great extension of my tutorial.

    Reply

Trackbacks for this post

1. Setup a Django VM with Vagrant, VirtualBox, and Chef | smalley creative blog
2. Managing Scripts and Configurations on Github with a script « Linux Tidbits
3. Sync dotfiles across multiple computers using Git & Github Pedram Navid
4. One git repo to rule them all! | Sebastians Blog

Got something to say? Go for it!

Name (required)

Mail (will not be published) (required)

Website

Submit Comment

← Spotlight – namebench

Tip: Easily Find Files In The Current Directory →

SMALLEY CREATIVE BLOG

Search for: To search type and hit Search

Help Out!

If you've found The Smalley Creative Blog useful, please give whatever you think would be fair to help pay for my time and hosting fees. Thank you!

Michael is the creator and sole maintainer of the popular **Smalley Creative Blog**, a regularly updated source of tutorials, news, and solutions of a technical nature.

He is interested in (in an ever-changing order):

entrepreneurship

devops, systems administration, monitoring

private cloud computing infrastructure

vintage computing, gaming, and pixel art (yes, pixel art)

road bicycling

teaching

traveling

musicianship

functional design

Recent Posts

- Automating Your OS X Deploy – Part I
- How To Disable Java on OS X Lion to Protect from Zero-Day Bug
- Amiga Emulation – An E-UAE Binary for OS X 10.7
- Fixing OpenLDAP Authentication on OS X Lion
- Setup a Django VM with Vagrant, VirtualBox, and Chef

Archives

- January 2013
- July 2012
- April 2012
- March 2012
- January 2012
- May 2011
- April 2011
- February 2011
- January 2011
- December 2010
- October 2010

Blogroll

- Claude's Internet Finds
- Musings of an Anonymous Geek
- Standalone Sysadmin
- SuperK.org
- The James Venuto Experience

Links

- Visit My LinkedIn Profile
- Visit the Smalley Creative Website

Latest Tweets

Warning: file_get_contents(http://search.twitter.com/search.atom?q=from%3Amichaeljsmalley&rpp=5) [function.file-get-contents]: failed to open stream: HTTP request failed! HTTP/1.0 410 Gone in **/home/smalleycreative/blog.smalleycreative.com/wp-content/themes/premiumpixels/admin/theme-functions.php** on line **143**

- RT @holman: I'm getting ready for seven years from now where everything will be thick serifs, opaque squares, and textures everywhere. 2 months ago
- RT @mattlinderman: I'm ok with more deaths due to terrorism if it means we maintain our civil

liberties. I think more of us need to say tha… 2 months ago

- Did Apple seriously just demo a password filler feature that SHOWS THE PLAIN TEXT PASSWORD ON SCREEN!???????? #wwdc #RAGEQUIT #NERDRAGE 2 months ago
- I'm holding out for OS X Compton. @apple @wwdc 2 months ago
- Apple today announced that they're dropping big cat names, and moving to California-themed names. Thanks for reminding me I don't live there 2 months ago

Follow @michaeljsmalley on Twitter