Using Git and GitHub to Sync Config Files between Machines

Posted At : November 4, 2009 11:39 AM | Posted By : Bob Silverberg
Related Categories: Git, OS X

This post is a follow-up to my earlier post about Placing Config Files Under Version Control with Git and GitHub. In that post I discussed how one can use Git and GitHub to place your config files under version control (via Git), and to maintain a backup of them (via GitHub). In this post I'm going to discuss a set up that will allow another machine's config files to stay in sync with the originals. The scenario I'm discussing involves config files, but one could use this approach for any set of files that one wants to keep in sync between two machines.

Getting a Copy of the Repo onto Machine 2

Assuming that you already have Git installed on Machine 2, follow these steps:

1. Open a Terminal window.

2. Change to a directory in which you want to put your Git repo. E.g.,

```
cd ~/gitRepos
```

3. Clone your GitHub repo to this machine:

```
git clone git@github.com:bobsilverberg/dotfiles.git
```

Note that the last part of that command, the one that says "git@github.com:bobsilverberg/dotfiles.git" is the same as the one listed as *Your Clone URL* on the GitHub home page for your project. You now have a local Git repo that is linked to the original Git repo on GitHub.

4. Just as you did on machine 1 (in the previous post), you'll have to create a symbolic link from each file in your Git repo to the location it is expected to be. E.g.,

```
ln -s -i -v ~/gitRepos/dotfiles/bash_profile ~/.bash_profile
```

Grabbing Changes from the GitHub Repo

Let's say you've made some changes to your config files on Machine 1, committed them, and pushed them to the repo on GitHub. Updating Machine 2 to reflect those changes is as easy as:

```
cd ~/gitRepos/dotfiles

git pull origin master
```

It Goes Both Ways

Let's say you're still working on Machine 2 and you decide that you want to make a change to your .bash_profile and have it reflected on both machines. Simply make the change and then:

```
cd ~/gitRepos/dotfiles

git add .

git commit -m 'updating .bash_profile to make the prompt prettier'

git push origin master
```

Of course you'd need to pull those changes when you got back to Machine 1. You could easily set up an alias to make the add/commit/push and pull simpler. For example, you might add the following entries to your .bash_profile:

```
alias dtup="cd ~/gitRepos/dotfiles; git add .; git commit -m 'updating dotfiles'; git push origin master"

alias dtdn="cd ~/gitRepos/dotfiles; git pull origin master"
```

Note that, unlike what you see displayed above, the code for each alias must be on a single line. With those aliases in your .bash_profile, you can simply type "dtup" to commit changes to your dotfiles to your local repo and push them to GitHub, and you can type "dtdn" to pull changes from GitHub into your local repo.

As I'm just getting started with Git, I'm not entirely sure yet when you need to specify the remote and branch (the *origin* and *master* in the examples above) and when you can leave them out. I've seen situations in which I can just use "git push" or "git pull" and it works, but other situations in which I have to specify the remote and branch. More on this as I figure it out.