

A Short (and probably not too good) Guide to Git

Author: Nick Charlton (<http://nickcharlton.net>)

Date: 21st February 2011

This is a quick, short guide to using Git. Consider this a pick of the top few important commands that you will end up relying on.

Creating a Repository

First you need to build out the basics of a directory structure. I currently have a 'git' directory in my user directory. Under which I have folders for each of my projects. So, here using "test" as an example:

```
$ mkdir ~/git
$ mkdir ~/git/test
$ cd ~/git/test
$ git init
```

Adding files

Here is an example of adding a README file. Using this example, you can also format your README's for use on GitHub, like as Markdown.

```
$ touch README
$ git add README
```

Obviously, it would help if your README file contained data. But I digress.

Committing Changes

Committing is as simple as this:

```
$ git commit -m 'My Commit Note'
```

I have generally used `git commit -am 'My Commit Note'` as this picks up all of the files I have been working on, tells me what it is doing and adds a commit note.

Undoing Changes You're Yet to Commit

Occasionally, when you're working on something you will want to reset everything in your working directory and start with how the repository looked on your last commit. To do this, enter:

```
$ git reset --hard HEAD
```

Pushing to GitHub

Once you have committed something to your repository, it's time to push it to GitHub. The following is taken from the guide GitHub provides once you create a repository.

```
$ git remote add origin git@github.com:nickcharlton/test2.git
$ git push origin master
```

These commands add somewhere for you to push the code, and then a branch to place it into.

Once you have done this a few times, you can get away with the simpler:

```
$ git push
```

Pulling

Pulling is the process of getting the code from the remote repository. In this case, your GitHub repo. This is done with the following command. It's much like **update** in Subversion.

```
$ git pull
```

Branching

The great flexibility of git is through it's power to create, merge and delete branches.

Branches allow you to take your current code base, do something different to it and then merge it back into the main trunk. The idea is to use a branch to try out something new and if it works, you'd merge it into the main part of the repository.

```
$ git branch
```

Lists all of your current branches.

```
$ git branch experimental
```

Would create a new branch called experimental with the contents of your current codebase. Then, issuing `git checkout experimental` will switch you to the new branch.

```
$ git checkout experimental
```

When you are in your new branch, you can commit as usual, however your changes won't have any effect upon your 'master' branch.

Deleting a Branch

```
$ git branch -D experimental
```

Fetching

fetch pulls down a remote repository.

```
$ git fetch experimental
```

This won't however integrate the pulled repository into your **HEAD**, you'll need to merge with **HEAD** to get there.

Merging

Merging is combining two branches.

This is done by calling a 'merge' in the branch you want to merge with.

```
$ git merge experimental
```

For example, whilst in the master branch, merging the experimental branch will pull in the code and make it live in harmony with your previous work.

Or, it won't and you'll end up with a merge conflict.

See: http://book.git-scm.com/3_basic_branching_and_merging.html

Syncing with a Remote Repository

If you have a repository being edited by others on a remote location, such as GitHub, to sync with your current branch, you can either run `git pull` or `git fetch` followed by `git merge`, the documentation suggests doing the latter for flexibility.

```
$ git fetch git@github.com: [user]/[repo].git  
$ git merge origin/master
```

Where [user] and [repo] represent the repository URL, and in the second line, you are merging with your current branch (this is by default), then the branch for which you pulled down. With GitHub, this is `origin/`, then the name of the repository, so for the main repo, `master`.

Tagging

Tagging is used when a specific bit of code reaches a certain milestone. Such as a version number. It's especially useful for packaging up releases, and for keeping issues tacked to a specific version.

This tags the last commit, with the name "v1.0".

```
$ git tag -a v1.0
```