# How To Git

Matt McCarthy

12 March, 2016

## 1 Version Control

## 2 Git 101

To start, we need to configure `git`. To do so, we run the following two commands.

```
git config --global user.name "Your Name Here"
git config --global user.email "your.email@host.domain"
```

These commands tell our `git` installation who we are and how to contact us so that other users know who is responsible for each commit.

### 2.1 Local Git

Our first task is to make a repository on our local machine. Next we want a directory in which we will store our `git` repositories. For the sake of simplicity, let's just make a new folder in the home directory called `git` (e.g. run `mkdir ~/git`). We now need to `cd` into our new directory, so we run `cd ~/git`. Now run `mkdir my-git-repo` and then `cd my-git-repo`. We will now turn this folder into a `git` repository by running `git init`. And now we have a `git` repository.

We're now going to start making changes, tracking them, and committing them. Let's begin by creating a file and telling `git` to track it. Run `touch file.txt`, this will create a file called `file.txt`. If we run `git status`, it will list `file.txt` as an untracked file. We now need to run `git add .`. The previous command tracks all untracked files and tracks any changes you made. If we run `git status` again, we will see that `new file: file.txt` is in the list of changes to be committed. Lastly, to commit our changes, we run `git commit -m "Added file.txt"`. This logs our changes and gives us a point to which we can revert. If we run `git status` once more, it will report that there is nothing to commit and that the working directory is clean. The `git add .` and `git commit -m "message here"` commands define the workflow on a single machine, that is these commands track and log each change you make to your project.

Next we'll make some changes to the project and then reset them. For now, run the following command.

```
echo "Subversion is the best version control software" > file.txt
```

Then track the change using `git add .` but don't commit the change just yet. Run `git status` and ensure that the change is tracked. You see, the statement we piped into `file.txt` is a lie and so we need to undo the change even though we just tracked it. To do so, we run `git reset --hard`. Now run `git status` to make sure the change is gone. More generally, we can run `git reset --hard commit-id` to reset to any given commit. Since the old statement is gone, we will go ahead and pipe the correct statement into `file.txt`.

```
echo "git is the best version control software" > file.txt
```

We will now track and commit our changes in one command by running `git commit -am "Did a thing"`. This command adds all changes in *tracked* files and then commits them in one fell swoop. Furthermore, it does not add any untracked files, so any new files would be skipped by this command.

### 2.2 Using GitHub

Now that we know how to deal with changes in a project locally, we're going to bring GitHub into the picture. To begin, if you do not have a GitHub account create one at `github.com`. Continuing with our toy project, create a repository on your GitHub account by clicking the 'New repository' button. For the name, call it `my-git-repo` and then create the repository. This should bring you to webpage of your new repository. From here copy the URL next to the 'Download ZIP' button. Our next step is to add the GitHub repository as a remote server. To do so run `git remote add origin copied-url`. Now that we have added the server, run `git push origin master` to push your changes to GitHub.

Now that your project is on GitHub run `cd ..` to go into the parent directory and then delete your project using `rm -rf my-git-repo`. We will now *clone* the repository from the GitHub URL. Now run `git clone copied-url`. This will create a directory for your project and then copy all of the project's files into that directory. To check run `cd my-git-repo` and then `cat file.txt` and it should output the contents of the file.

When we use a `git` server, the workflow will change a little bit. Instead of just running `git add` and `git commit`, we now have to run `git push` in order to push our commits to the server. Furthermore, instead of running `git init`, we can just create an empty repository on the server and then use `git clone` to set it up on our machine.

Lastly, GitHub has a Git cheat sheet at

```
https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf
```

if you wish to have a quick reference document.

## 3 Git Demonstration

### 3.1 Forking the Demo

In your web browser, navigate to `https://github.com/matt-mccarthy/cnu-foss-day-demo`. Once the page has loaded look for a button called 'Fork' and click it in order to fork the repository. After that you should be on the page for your forked repository. From here, clone your repository and then in your terminal use `cd` to enter the repository's directory. Once you have `cd`'d into the repository, run the following command.

```
git remote add upstream https://github.com/matt-mccarthy/cnu-foss-day-demo.git
```

This will allow you to pull changes from the original repository into your own.

### 3.2 Claiming and Fixing an Issue

### 3.3 Pull Changes from Upstream

### 3.4 Pull Requests

## 4 Advanced Git

### 4.1 Branches

### 4.2 Tags

### 4.3 .gitignore