Git & GitHub | Git

Introduction

Let's start by creating a directory to play with Git.

```
$ mkdir -p ~/code/prework/git-practice
$ cd ~/code/prework/git-practice
$ touch data.txt
```

Creating a Repository

```
$ git init
```

Let's run an ls command to show hidden folders and files inside of the folder:

```
$ ls -a
```

In addition to data.txt, we have a new folder called .git. We did not create this directory ourselves, Git did when we ran the git init command.

```
$ ls -a .git
```

The .git repository is where Git keeps track of all of the changes you make, and much more. There's no point in going into the specific details of the folder now, but we will reference it as the lesson progresses.

```
$ rm -rf .git
```

Be careful though. rm -rf is a dangerous command that can erase everything on your system if used improperly. In addition, any history you may have in the git repo will be gone.

Where to Create a Respository

Most of the time, you will want to create the Git repository in a specific project folder. You do not want to create a Git repository in a high level folder, such as Documents, or your home(~) directory.

Why? Git keeps track of the folder, and all subfolders of that folder. This means that if you create a Git repository in your home(~) directory, you will be tracking all of the changes to files and folders on your local user's computer.

Git status tells us what files and folders are being tracked, and what their current status is according to Git.

```
$ git status

Staging a Single File


$ git add data.txt
$ git status
```

Staging Multiple Files

```
$ touch file1.txt file2.txt file3.txt file4.txt file5.txt
$ git status
$ git add file1.txt file2.txt
$ git status
```

On second thought, I'd like to stage all of the files in my project:

```
$ git add -A # -A stands for "All"
$ git status
```

Un-staging Files

```
$ git reset file5.txt
$ git status
```

Saving Changes:

We've already added files to our staging area from our working directory, now we need to move them permanently to our repository. The state of our project will forever be frozen in that snapshot.

```
$ git commit -m "Initial commit - Added data, and created needed files"
$ git status
```

Revisiting and Viewing Commits:

At some point in the future, it is likely we'll want to view all of the changes to our project. In addition, we may want to revert our project to a previous snapshot.

The git log command is used to view commits, and data about those commits.

```
$ git add file5.txt
$ git commit -m "Add file5 to repo"
$ git log
```

Creating a GitHub Repo

Click the plus sign in the upper right hand corner of the page, and then click "New Repository"

Adding a Remote Repository:

The first step in this is connecting the two repos. This can be done using the git remote add command, while supplying a few options:

Alias. You create an alias in your system for the remote repository, which will be pointing to the GitHub repo. It's very common to call this Alias origin, but you can use the name you want.

GitHub repository URL. Unique URL that GitHub provides for each of your repositories.

Navigate to your git-practice folder on your computer if you're not already there, and run the following:

```
$ git remote add github-repo https://github.com/<your-github-username-her
e>/git-practice.git
```

This is telling your local GitHub repo: "Add a remote repository called github-repo, and have it point to https://github.com/jalexy12/git-practice.git".

We can view a list of remote repos attached to our current local repository by running git remote with no options:

```
$ git remote
```

Pushing to a Remote Repository:

Pushing to a remote repository is pretty simple in most situations. We use the git push command, along with a couple of options to push our local repository to our remote repository:

```
$ git push github-repo master
```

Pulling From a Remote Repository:

Occasionally GitHub repos will have changes that we don't have on our local computer. This can happen when teammates push code to the repo, or when we update code on GitHub manually.

```
$ git pull github-repo master
```

How to Fork

Forking a repository is taking someone else's code, and creating your own copy of it. Eventually, the changes you make in the fork may be included in the original repository. On GitHub this simply means taking someone's remote repo, and copying it to your own remote repo.

Luckily, forking on it's own is super simple. Navigate to this repo and click the fork button in the upper-right corner of the page. Choose your own account as the place to fork.

How to Clone

Cloning a repository is simply taking a remote repository and copying it to your local machine.

Now that we've forked our own version of the exercise repo, we need to copy it to our computer so we can work on it. On your own copy of the repository, click the Copy or Download button, and copy the link to your clipboard.

```
$ cd ~/code # Make this directory if it does not exist
$ git clone < link you just copied to your clipboard >
```

Cloned repositories already have a Git repo in it. You do not need to run git init again.