Introduction to Git & Github

Step 0: Install git and create a GitHub account

The first two things you'll want to do are install git and create a free GitHub account.

Follow the instructions here to install git (if it's not already installed). Note that for this tutorial we will be using git on the command line only. While there are some great git GUIs (graphical user interfaces), I think it's easier to learn git using git-specific commands first and then to try out a git GUI once you're more comfortable with the command.

Once you've done that, create a GitHub account here. (Accounts are free for public repositories, but there's a charge for private repositories.)

Step 1: Create a local git repository

When creating a new project on your local machine using git, you'll first create a new repository (or often, 'repo', for short).

To use git we'll be using the terminal.

To begin, open up a terminal and move to where you want to place the project on your local machine using the cd (change directory) command. For example, if you have a 'projects' folder on your desktop, you'd do something like:

```
mmhanna:Desktop mmhanna$ cd ~/Desktop
mmhanna:Desktop mmhanna$ mkdir myproject
mmhanna:Desktop mmhanna$ cd myproject/
```

To initialize a git repository in the root of the folder, run the `git init` command:

```
mmhanna:myproject mmhanna$ git init
Initialized empty Git repository in /Users/mmhanna/Desktop/myproject/.git/
```

Step 2: Add a new file to the repo

Go ahead and add a new file to the project, using any text editor you like or running a `touch` command.

Once you've added or modified files in a folder containing a git repo, git will notice that changes have been made inside the repo. But, git won't officially keep track of the file (that is, put it in a commit - we'll talk more about commits next) unless you explicitly tell it to.

```
mmhanna:myproject mmhanna$ touch mmhanna.txt
mmhanna:myproject mmhanna$ ls
mmhanna.txt
```

After creating the new file, you can use the `git status` command to see which files git knows exist.

```
mmhanna:myproject mmhanna$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    mmhanna.txt

nothing added to commit but untracked files present (use "git add" to track)
```

What this basically says is, "Hey, we noticed you created a new file called mmhanna.txt, but unless you use the `git add` command we aren't going to do anything with it."

An interlude: The staging environment, the commit, and you

One of the most confusing parts when you're first learning git is the concept of the staging environment and how it relates to a commit.

A commit is a record of what files you have changed since the last time you made a commit. Essentially, you make changes to your repo (for example, adding a file or modifying one) and then tell git to put those files into a commit.

Commits make up the essence of your project and allow you to go back to the state of a project at any point.

So, how do you tell git which files to put into a commit? This is where the staging environment or index come in. As seen in Step 2, when you make changes to your repo, git notices that a file has changed but won't do anything with it (like adding it in a commit).

To add a file to a commit, you first need to add it to the staging environment. To do this, you can use the git add command (see Step 3 below).

Once you've used the git add command to add all the files you want to the staging environment, you can then tell git to package them into a commit using the git commit command.

Note: The staging environment, also called 'staging', is the new preferred term for this, but you can also see it referred to as the 'index'.

Step 3: Add a file to the staging environment

Add a file to the staging environment using the git add command.

```
git add mmhanna.txt
```

If you rerun the git status command, you'll see that git has added the file to the staging environment (notice the "Changes to be committed" line).

```
mmhanna:myproject mmhanna$ git status
On branch master

Initial commit
```

```
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   mmhanna.txt
```

To reiterate, the file has not yet been added to a commit, but it's about to be.

Step 4: Create a commit

It's time to create your first commit!

Run the command git commit -m "Your message about the commit"

```
mmhanna:myproject mmhanna$ git commit -m "This is my first commit!"
[master (root-commit) b345d9a] This is my first commit!
 1 file changed, 1 insertion(+)
 create mode 100644 mmhanna.txt
```

The message at the end of the commit should be something related to what the commit contains - maybe it's a new feature, maybe it's a bug fix, maybe it's just fixing a typo. Don't put a message like "asdfadsf" or "foobar". That makes the other people who see your commit sad. Very, very, sad.

Step 5: Create a new branch

Now that you've made a new commit, let's try something a little more advanced.

Say you want to make a new feature but are worried about making changes to the main project while developing the feature. This is where git branches come in.

Branches allow you to move back and forth between 'states' of a project. For instance, if you want to add a new page to your website you can create a new branch just for that page without affecting the main part of the project. Once you're done with the page, you can merge your changes from your branch into the primary branch. When you create a new branch, Git keeps track of which commit your branch 'branched' off of, so it knows the history behind all the files.

Let's say you are on the primary branch and want to create a new branch to develop your web page. Here's what you'll do: Run git checkout -b . This command will automatically create a new branch and then 'check you out' on it, meaning git will move you to that branch, off of the primary branch.

```
git checkout -b my-new-branch
```

After running the above command, you can use the git branch command to confirm that your branch was created:

```
mmhanna:myproject mmhanna$ git branch
  master
* my-new-branch
```

The branch name with the asterisk next to it indicates which branch you're pointed to at that given time.

Now, if you switch back to the primary branch and make some more commits, your new branch won't see any of those changes until you merge those changes onto your new branch.

Step 6: Create a new repository on GitHub

If you only want to keep track of your code locally, you don't need to use GitHub. But if you want to work with a team, you can use GitHub to collaboratively modify the project's code.

To create a new repo on GitHub, log in and go to the GitHub home page. You should see a green '+ New repository' button:

After clicking the button, GitHub will ask you to name your repo and provide a brief description:

When you're done filling out the information, press the 'Create repository' button to make your new repo.

GitHub will ask if you want to create a new repo from scratch or if you want to add a repo you have created locally. In this case, since we've already created a new repo locally, we want to push that onto GitHub so follow the '....or push an existing repository from the command line' section:

```
mmhanna:myproject mmhanna$ git remote add origin https://github.com/mmhanna/mynewrepository.git
mmhanna:myproject mmhanna$ git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 263 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/mmhanna/mynewrepository.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

(You'll want to change the URL in the first command line to what GitHub lists in this section since your GitHub username and repo name are different.)

Step 7: Push a branch to GitHub

Now we'll push the commit in your branch to your new GitHub repo. This allows other people to see the changes you've made. If they're approved by the repository's owner, the changes can then be merged into the primary branch.

To push changes onto a new branch on GitHub, you'll want to run git push origin yourbranchname. GitHub will automatically create the branch for you on the remote repository:

```
mmhanna:myproject mmhanna$ git push origin my-new-branch
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 313 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/mmhanna/mynewrepository.git
 * [new branch]      my-new-branch -> my-new-branch
```

You might be wondering what that "origin" word means in the command above. What happens is that when you clone a remote repository to your local machine, git creates an alias for you. In nearly all cases this alias is called "origin." It's essentially shorthand for the remote repository's URL. So, to push your changes to the remote repository, you could've used either the command: git push git@github.com:git/git.git yourbranchname or git push origin yourbranchname

(If this is your first time using GitHub locally, it might prompt you to log in with your GitHub username and password.)

If you refresh the GitHub page, you'll see note saying a branch with your name has just been pushed into the repository. You can also click the 'branches' link to see your branch listed there.

Step 8: Get changes on GitHub back to your computer

Someone else could be collaborating with you on your git repo, and the repo on GitHub may look a little different than what you have on your local machine.

In order to get the most recent changes that you or others have merged on GitHub, use the git pull origin master command (when working on the primary branch).

```
mmhanna:myproject mmhanna$ git pull origin master
From https://github.com/maggiemhanna/my-repository
 * branch            master     -> FETCH_HEAD
Already up to date.
```

This shows you all the files that have changed locally and how they've changed. Since there's no difference between the local and the github repo, it will show you a message: Already up to date.

Now we can use the git log command again to see all new commits.

(You may need to switch branches back to the primary branch. You can do that using the git checkout master command.)

```
mmhanna:myproject mmhanna$ git log
```