Initial Set Up

Note: *At some point during this process you may asked on the terminal to sign-in. Don't panic, just sign in using your GitHub username and password.*

After setting up your GitHub as detailed in the README navigate to your repositories (repos) and click "New". Follow the prompts and name your project to match the one on your computer and create your repo.

Open the terminal and cd to the desired project. *Note: everything in capital letters and square brackets should be replaced.*

Enter this command to initialize git in the project

```
git init
```

Go back to your browser and the link to your project -it's under "Quick Setup". *Note: Use the HTTPS link not the SSH link for our purposes.* It should look something like this:
https://github.com/[YOUR GITHUB USERNAME]/[NAME OF YOUR PROJECT].git

Switch back to the terminal and enter the command below. This connects the project on your computer to the one on GitHub.

```
git remote add origin [LINK TO PROJECT]
```

This command stages all of your files for committing

```
git add --all
```

This command essentially saves the changes you've made to your files locally (on your computer). Since this is the first commit it simply saves the files. It's a good practice to leave a descriptive message in quotes so you can trace back through changes you've made in your project.

```
git commit -m "first commit"
```

This command pushes the project to remote repo (you can now see it on GitHub basically)

```
git push origin master
```

Useful git commands:

This command shows you what you're on, what files are staged for committing, if there have been any changes, et cetera. SUPER USEFUL.

```
git status
```

This command enables you to add individual files for commits

```
git add [FILE NAME]
```

Using Different Branches

Before making any changes it a good habit to "pull" any changes to your local repo using the command below. Typically, you'd use "master" in the space for the branch name unless you specifically wanted to pull from a specific development branch.

```
git pull origin [BRANCH-NAME]
```

This command creates a new branch that you can edit your project in. Think of your master branch like a final product.

```
git checkout -b [BRANCH-NAME]
```

This command allows you to switch between branches including the master branch. *Note: -b is not present in this command.*

```
git checkout [BRANCH-NAME]
```

This command allows you to delete a branch

```
git branch -d [BRANCH-NAME]
```

This command allows to push to a branch that is not the master

```
git push origin [BRANCH-NAME]
```

These commands merge a branch with the master branch

```
git checkout master
git merge [BRANCH-NAME]
```

Use this command to change the message your previous commit

```
git commit --amend -m "[FIXED MESSAGE]"
```

OR

```
git commit --amend
```

Add your message and type `:wq` to get out of the message space (can't think of a better word for atm)

Commands to use when you've done something weird and need to figure out how to fix it

git stash

This command temporarily "stashes" changes you've made

```
git stash
```

This command reapplies the changes you've made

```
git stash pop
```

You can create multiple stashes I would recommend leaving a message that describes what your stashing when initially creating the stash. If you're doing it once, go ahead and use the commands above.

```
git stash save "[MESSAGE ABOUT CHANGE THAT YOUR STASHING]"
```

This command shows you what you have stashed.

```
git stash list
```

When you enter the command above, the stashes will appear like this:

```
stash@{[STASH NUMBER]}
```

This command deletes a specific stash

```
git stash pop stash@{[STASH NUMBER]}
```

git diff and git log

These commands compares two branches. To exit out of this press "q"

```
git diff [BRANCH A] [BRANCH B]
git diff [BRANCH A]..[BRANCH B]
git diff [BRANCH A]...[BRANCH B]
```

The commands with the double and without dots are the same and will show you the difference between the tips (last changes) of the two branches. The command with the triple-dot will show you the difference between the common ancestor (where the branches started to differ) of the two branches and the tip branch B while ignoring the changes made to branch A.

This command shows what commits have been made

```
git log [BRANCH-NAME]
git log [BRANCH 1]..[BRANCH A]
git log [BRANCH 1]...[BRANCH B]
```

For single branch there aren't any comparisons. The double-dot shows the commits that are in B but not in A. The triple-dot shows the commits that are in A but not in B AND the commits that are in B but not in A.

Reset

DANGER: This command deletes all of your local changes

```
git reset --hard
```

Useful links

https://git-scm.com/docs
https://dangitgit.com/
https://www.atlassian.com/git/tutorials