Git Init

A lab about git. Written by an TA who wanted to have fun while writing the lab instructions.

You can download git from here if you want it on your personal computers.

https://git-scm.com/

Purpose

Git is a version control system (VSC) that is very common in industry today. It is really just a fancy way to save your work. Git allows multiple people to work on the same project simultaneously from their own computers and it saves revisions of projects in case users want to start something over. Git has a bit of a learning curve but this lab will help you get started.

How Git Works

A git repository (repo) is the highest level of organization in git. If you are starting a new project, you will usually make a new repo for it. Think of these repos as one big folder. Inside the folder you can add files or make subfolders. For our purpose, we will just say these live externally on the internet.

In order to work with these repos, you need to make a copy of it on the machine you will be working on. This is called cloning a repo. Once you have cloned a repo, you can access it through your regular file system. Because you are working with a copy, only you can see the stuff you add. You are making changes to the local copy, not the external copy. In order to make changes to the external copy, you have to stage your changes, commit your changes, and then push your changes.

Staging your changes means you are flagging them to be saved later by a commit. When you commit your changes, everything you had in staging gets saved. Think of staging as putting a copy of things into a box and committing as closing the box so no more stuff can be put in.

Once you have a commit (a box) you want to send it to the external repo. This is what pushing is for. Pushing sends your local commits to the external repo.

What if multiple people shared the same external repo? When they cloned the repo, they would all be at the same starting place but after the first person pushes, how to the others catch up? They could clone the whole repo again but that would copy every file again and it could take a long time for really big repos. Instead they can pull just the changes between their local repo and the external repo. Pulling doesn’t erase the new work you have added, it just updates your copy with other people’s commits.

There are some other tricks to learn about git, such as branching, but we won’t worry about those in this class.

Git Commands

We will be using command line git in this lab. Third party software is available to provide a visual experience for git but third party software is not always installed on the computer you’ll be working on. If a computer has git installed, the command line will always be the same.

These are commands that you will use a lot. For a complete list, google it ;)

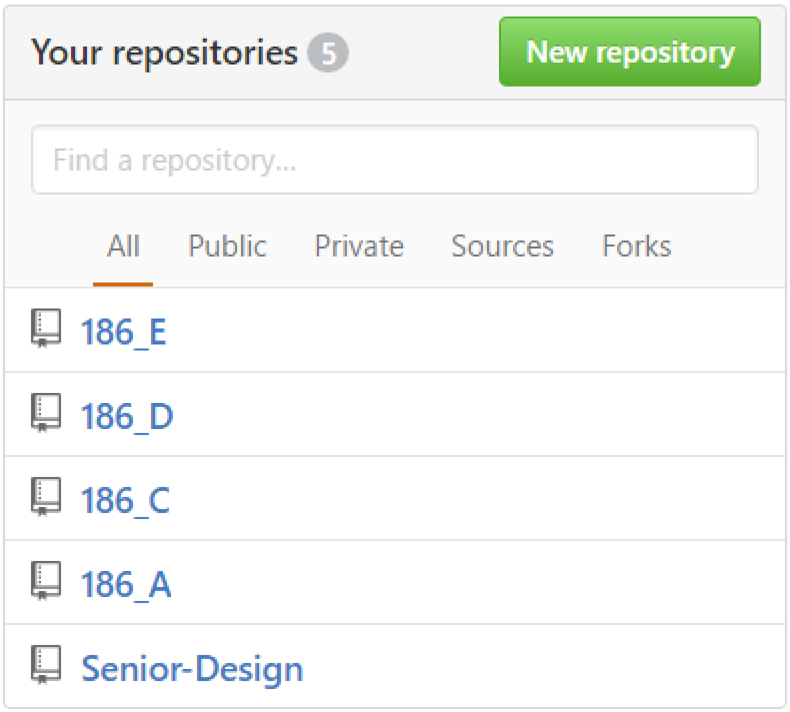
- **git clone** or **git init** – this is how you start a git repository. You’ll usually only need this once per project.
- **git pull** – this is how you update your local copy.
- **git add** – this is how you stage new files or changes that you want to make. This doesn’t save them, it flags the files to be saved later.
- **git status** – this is how you see the status of the staging area
- **git commit** – this is what actually saves the changes in git on your local machine.
- **git push** – this is how you share your local changes with other people in your project.

Enough Reading, onto the Actual Lab

The colors and domain of command line screenshots may not match what you see. However, the commands are still the same.

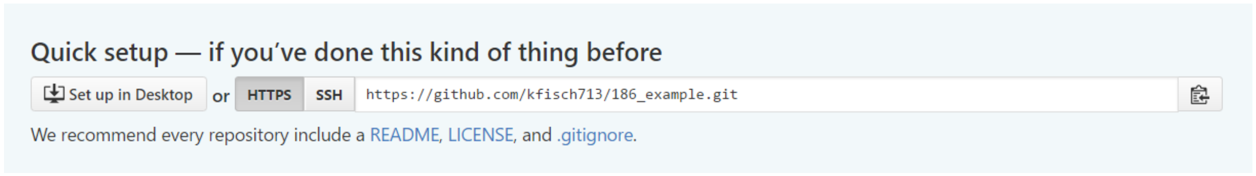
1. We are going to use github as our mysterious place on the internet for our external repos. Go make a github account. https://github.com/. You can use your iastate email address or a personal email address. It is recommended to use a personal email address so that you can use this github account after you graduate but it is not required.

2. Make a new repository on github. There are multiple ways to do this and many of them involve a green button shown below. Name it whatever you would like.



3. Open git bash and switch to the directory you would like to be in. Press the windows key and type "git bash" to search for the program.

4. Clone the repo you made using the command **git clone <address>**. You will need the https address of your repo from github. Look for the quick setup box or copy the url from your browser. Make sure it has ".git" at the end.



```
https://github.com/kfisch713/186_example.git
```

We recommend every repository include a README, LICENSE, and .gitignore.

Example: **git clone https://github.com/kfisch713/186_example.git**

5. Download the broken code from BlackBoard and fix it. Save it to your cloned git repo.

6. Add the fixed files from BlackBoard and a file of your choosing to the staging area. Use the command **git add <name>** and **git status** to add the files you want. Try to remove a file from staging after it has been added. Ask Professor Google how to do this.

Note
You don't typically want to add compiled code to a commit. Every machine is a little bit different so you usually want to compile your own code.

7. Commit your code using **git commit –m "<message>"**. The –m and message part are important. The –m signifies you want to add a commit message. The <message> part can be replaced with whatever commit message you would like. The quotes around it are necessary. If you choose to just use **git commit** then you will be taken to a vim style screen and asked to put in your message there.

8. Almost done, now push your commit to the external repo. Use **git push origin** for your first push. Depending on how you use git for a project, pushing directly to origin can be dangerous but it is okay to do here. This will create some links behind the scenes that are necessary. After your very first push to origin, you should only have to use **git push**. You can try adding a new file, committing it, and pushing it using just **git push**.

   You may be prompted for you github username and password. This is okay. There are ways to setup git so that you aren't prompted every time but it is not necessary for this lab.

9. If needed, use **git config –global push.default simple**.

10. If you didn't get any errors, go refresh the webpage with your github. You should see your files! Call over the TA and show them the fixed code on github for your grade.