Introduction to git

Part 1: Setting up and cloning

- If you don't have a github account, create one and login
- If you are new to github or haven't in the past upload your ssh key.
- The base repository that you should use for this class is here
    - Next you should fork (make your own version of) this repo. You can do this by hitting the fork button near the top of the screen on the right
- Add code-cullison as a collaborator
    - Looking at your repository is how your labs will be graded
- Install git on your local computer if you don't have it installed
- Make a copy of this repository on your local machine by "cloning" the repository. You should see a clone button near the top on the right that gives you the link. Use the ssh version.More information can be found here
- If you have never run git on your machine you need to set your username and email. Take a look here on how to do this and other useful variables to set.

You should now have a copy of the repository on your local machine and are ready to proceed!

Part 2: The basics

At the simplest level git keeps a record of changes to your repository. If you look at the directory you cloned there are only two files in it README.md and prog.py. These are the only two files that are currently being tracked by git. You might (depending on your settings) also see a directory named .git. This is where git stores all of its information and except for a very few cases you should never modify anything in this directory.

Let's begin by adding another file to the repository. In your favorite text editor create a file named *.gitignore* in the directory. The *.gitignore* file server tells what git files to not to track by default. Take a look at this repo fill in your .gitignore with appropriate file patterns. For example if I work on the mac and use VS Code as my editor I would add the contents of VisualStudioCode.gitignore and macOS.gitignore.

Once you have created your .gitignore file tell git to start tracking the file. You do this by running

git add .gitignore
or

git add .

The first option adds just the .gitignore file the second will add all files that aren't tracked in your directory that don't follow a pattern in your .gitignore file.

Next let's commit changes to the git repo.

git commit -m "added .gitignore"

We have now made a new checkpoint to our repository. These changes only exist locally in the .git directory. We can send them to or github repository by pushing those changes to the cloud.

git push

At this stage if you reload your repository on github you should see the file. If there are changes in our github repository that we want to grab we can use the *git pull* command.

Part 3: Branching and merging

For an in depth understanding of branching take a look at this webpage or this video. I recommend that you take a look at least one of these sources unless you are very familiar with branching. One of the main uses of branches is to fix bugs.

If you have taken a look at prog.py you should quickly see some bugs. For this first part of the assignment I want you to create a branch to fix the bugs in the file. Begin by creating a new branch

git branch bugfix

We have now created this new branch locally. We can say we want to start working on this branch with

git checkout bugfix

In your favorite text editor fix the bugs in *prog.py*, and commit the changes. Next lets push our branch to our github repository.

git push origin bugfix

Finally lets merge our fixed program back to the main branch. We will begin by checking out the main branch and then merge the changes in bugfix back into our main branch.

git checkout main

git merge bugfix

Part 4: Submodules

Submodules provide a powerful way to handle the case where you depend on another repository. There are many use cases for submodules. One example is when you are writing a base library used by many different applications. In this case we are going to use it for the labs and any notebooks created in class.

Let's begin by adding to your repository notebooks done in class.

git submodule add http://zapad.stanford.edu/GP257/labs-2023/class-notebooks.git class-notebooks

After you have added the submodule make sure to run *add,commit,* and *push.* This will add to your repo the repo that will be used in class for the class notebooks. What git does is make a link to the **current** version of the repository. If the repository changes you will need to tell git to update the link to the current version using

```
git submodule update --init --recursive --remote
```

If for whatever reason you need to remove a submodule the procedure is

git submodule deinit class-notebooks

rm -rf .git/modules/class-notebook

git rm -rf class-notebooks

git add .

Notes

- That this is the first time we have done anything with the .git directory.
- We have introduced the rm command

- We have told git that we want to checkpoint this new version without the submodule

Part 5: Command line forking

This section is not required, but is suggested. Begin by download and installing github cli tools. Then login into your github account.

```
gh auth login --web
```

You can then fork the next lab by running

gh fork http://zapad.stanford.edu/GP257/labs-2023/basic-python-and-jupyter-notebooks.git

For the remainder of the term you should add your labs to your class git repo when you finish them