GISC 4317/6317: GeoComputation/Computer Programming For GIS

**Instructor: Dr. Bryan Chastain**

**Lab 12: Git**

OBJECTIVES

In this project, you will learn how version control can be used to keep track of what one person did and when.

Installing Git

* You will need to install Git on your computer for this lab. Depending on what type of machine you are using, there are different download options:
  + Windows: <https://git-scm.com/download/win>
    - If this is your home machine, choose “64-bit Git for Windows Setup”
    - If this is a lab machine or a machine where you don’t have administrative privileges, choose “64-bit Git for Windows Portable”
    - On either option, note the installation path, and click through the installation prompts, keeping all the defaults.
  + Mac
    - <https://git-scm.com/download/mac>
* To test that it installed correctly, we will need to run it from the command line.
  + Windows:
    - Open either Windows Power Shell or Windows Command Prompt
    - Type: git –version
    - If you are running Git Portable and this does not work, try searching for the “git-cmd” application in the folder that you installed it and running that. Then run the above command.
  + Mac:
    - Open the Terminal app
    - Type: git –version

Create github.com account

In order to push to github, you will need your own account.

* Go here: <https://github.com/join>
* You will need to enter your email address and then verify your account after registering.

**Starting with Git**

The first time we use Git on a new machine, we need to configure a few things:

*If using Git Portable in the lab, need to set the HOME environment variable first:*

*set HOME=%USERPROFILE%*

git config --global user.name "My Name"

git config --global user.email "my@email.com"

git config --global color.ui "auto"

Git commands are written git verb, where verb is what we actually want it to do. In this case, we're telling Git:

* our name and email address,
* to colorize output,
* that we want to use these settings globally (i.e., for every project),

The commands above only need to be run once: the flag --global tells Git to use the settings for every project on this machine.

**Starting with Git**

Once Git is configured, we can start using it to share code on [GitHub](https://github.com/). Make sure to create a GitHub account on the website if you don't yet have one.

Follow [these instructions](https://help.github.com/articles/creating-a-new-repository/) to create a new GitHub repository. Be sure to select these options:

* Call the repo "giscX317\_lab12"
* Create a README.md file
* Don't create a .gitignore file
* Don't add a license for now.

After you create your directory, clone a local copy onto your computer by following [these instructions](https://help.github.com/articles/cloning-a-repository/). Be sure to clone in a location that you will remember!

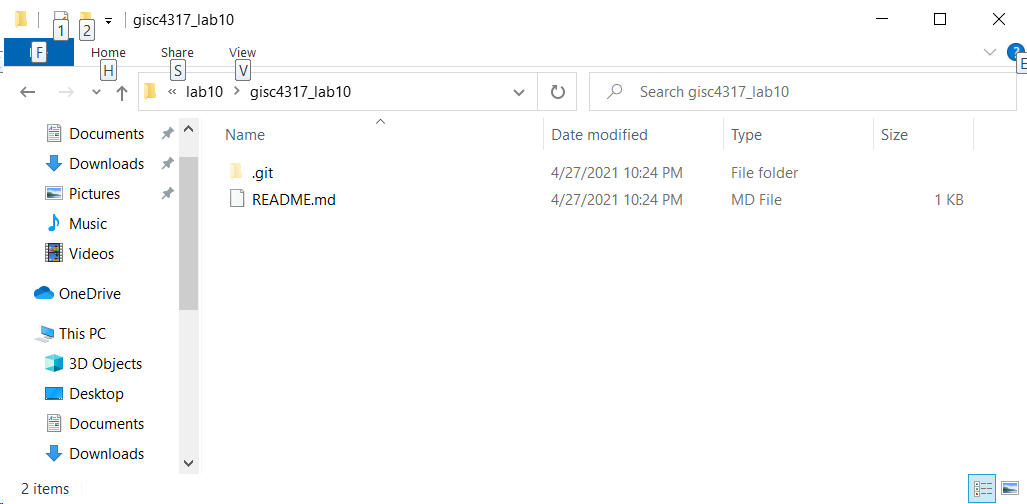
cd path/to/your/lab12/directory

git clone https://github.com/YOURUSERNAME/giscX317\_lab12.git

Now, navigate into your new git repository:

cd giscX317\_lab12

If you were to look in at the files in the folder with hidden folders displayed, it would look something like this:



Git stores information about the project in this special hidden .git sub-directory. If we ever delete it, we will lose the project's history.

We can check that everything is set up correctly by asking Git to tell us the status of our project:

git status

**git add – Tracks files**

Let’s add a file into our directory.

* On Windows, type: echo. > “file.txt”
* On Mac, type: nano file.txt

Now if we type: git status, we should get a message about untracked files, which is Git’s way of telling us there are files in the directory that Git isn’t keeping track of.

We can tell Git it should do so by using the add command:

git add file.txt

Or you could alternatively add all new files in the directory using the . shortcut:

git add .

Now using status will show there are no longer any untracked files.

**git commit – Saves files**

Git now knows that it's supposed to keep track of all the files in your repo, but it hasn't yet recorded any changes you've made to those files. To get it to do that, we need to run one more command:

git commit -am "First Commit"

When we run git commit, Git takes everything we have told it to save by using git add and stores a copy permanently inside the special .git directory. This permanent copy is called a revision and its short identifier is some random hexadecimal number.

We use the -a flag (for 'all') to tell Git that we want to commit all the changes we've made to every file. If we just run the git commit without the -a option, Git will expect us to specify which file's changes we want saved.

We use the -m flag (for "message") to record a comment that will help us remember later on what we did and why. If we just run git commit without the -m option, Git will launch nano (or whatever other editor we configured at the start) so that we can write a longer message.

If we run git status now it tells us everything is up-to-date. If we want to know what we’ve done recently, can track history using:

git log

This lists all revisions made to a repository in reverse chronological order. The listing for each revision includes the revision's full identifier (which starts with the same characters as the short identifier printed by the git commit command earlier), the revision's author, when it was created, and the log message Git was given when the revision was created.

**git push – Moves changes from one branch to another**

Systems like git allow us to move work between any two repositories. In practice, it's easiest to use one copy as a central hub, and to keep it on the web rather than on someone's laptop.

This is where GitHub comes in: it holds the master copy of a repository, and allows us to move changes between multiple local copies.

To copy our changes from our laptop to our GitHub repo, we can use git push:

git push -u origin main

Now open up a web browser and navigate to your GitHub repository. What do you see?

**Fetching/Syncing**

Oftentimes we need to sync our local repo with the master branch (the default branch) on GitHub. For instance, let's say you have two computers, one at home and one at work. We made a change using our work computer, and pushed it to the master branch on GitHub. But then we go home and find that our local copy is out of date.

The most common method of syncing branches is to use git fetch followed by git merge; or git pull.

* git fetch followed by git merge combines your local changes with changes made by others.
* git pull is a convenient shortcut for completing both git fetch and git merge in the same command

**Collaboration**

Version control really comes into its own when we begin to collaborate with other people.

There are two main ways to collaborate on github:

1. Adding individual collaborators to a project
2. The fork & pull model.

The first method adds users to your project, giving them full permissions to make changes. Collaborating in this fashion is very similar to the workflow described above.

**Fork & Pull**

GitHub also allows you to to accept individual contributions from users without granting them full access. This if referred to as the [Fork & Pull model](http://stackoverflow.com/questions/11582995/what-is-the-fork-pull-model-in-github).

Fork & Pull involves the following steps:

1. Fork an existing repo

The first step in in this workflow is to fork an existing repository. A [fork](https://help.github.com/articles/working-with-forks/) is a copy of a repository that you manage yourself. Forks let you make changes to a project without affecting the original repository.

To fork a repo:

1. On GitHub, navigate to [https://github.com/bchastain/gisc4317\_6317\_lab12](https://github.com/bchastain/gisc4317_6317_lab10)
2. In the top-right corner of the page, click Fork.

Now you have a fork of the original repo in your-user-name/gisc4317\_6317\_lab12

2. Commit a change

We've already seen how you can commit a change directly in GitHub's web interface. But when working with code, you often want to develop your scripts on your computer, so you can test it using Python, etc.

To do this, you first need to clone your fork onto your computer.

1. On GitHub, navigate to your fork of the gisc4317\_6317\_lab12 repository.
2. In the right sidebar of your fork's repository page, copy the clone URL for your fork.
3. Use git clone to clone the repo. Use a different path than your clone of the first repo above!

cd path/to/your/lab12/directory

git clone https://github.com/YOURUSERNAME/gisc4317\_6317\_lab12.git

We're now ready to make a change to the repo. Create a file in the 2022\_fall folder:

cd gisc4317\_6317\_lab12

* On Windows, type: echo. > "2022\_fall/FIRSTNAME\_LASTNAME.txt"
* On Mac, type: nano 2022\_fall/FIRSTNAME\_LASTNAME.txt

Open up that file in a text editor and add:

The title of your final project (in a header)

* A 1-2 sentence description of the project
* A link to your github repo (that you just made)

*Markdown Reminder*

Files with the extension .md are called markdown files. Markdown is a markup language used to convert plain text to HTML and many other formats. It's basically a way to add markup to a text (making things bold, lists, links, etc) using very simple syntax. It is often used in README files in software packages. You may have also noticed that all of the lessons for this course are written in markdown, as is many of the text files on GitHub. You can learn more about how to write GitHub-flavored markdown [here](https://help.github.com/articles/markdown-basics/).

Then add, commit, and push the change.

git status

git add 2022\_fall/FIRSTNAME\_LASTNAME.md

git commit -am "my final project info"

git push

3. Submit a pull request

Navigate to your GitHub repo (online) and check out your change!

Remember when you forked the repository originally? That means that your repository is different from mine, and from everybody elses. What if you want to share your change with others?

To do this, navigate to your GitHub repository and click the green icon to submit a pull request.

After you submit, I have the option to accept.

4. Keep your fork synced

It's good practice to keep your fork synced with the upstream (i.e. the original) repository. That way, if I make a change to gisc4317\_6317\_lab12, you can easily pull that change into your fork.

You can configure Git to pull changes from the original, or upstream, repository into the local clone of your fork.

git remote -v

git remote add upstream https://github.com/bchastain/gisc4317\_6317\_lab12.git

git remove -v

With git remote -v, you'll see the current configured remote repository for your fork.

Now you can sync your fork with the upsteam repo using git fetch:

git fetch upstream

git merge upstream/master

**Homework (Grad & Undergrad):**

Simply complete the in-class portion and once I have accepted your pull request, you will receive full credit.