Git Tutorial

brayden look

May 2023

1 What is Git?

Git is a way to track changes to files (**version control**). If you want to modify code and test changes without worrying about losing progress, Git is a more efficient alternative to saving sixteen different versions of a file that you keep losing track of.

Note that **Github is different than Git**; Git is a program that can perform version control, while Github is just a place to store and share those changes in a centralized way.

If you haven't used Git, it can be intimidating, or it can seem like one of those things that you "ought" to do, but that you never end up actually doing (like leaving comments on your code). Hopefully this document will make it easier to regularly use Git.

2 Installing Git

Warning: The instructions for installation are mostly specific to Windows. They are likely to be parallel to other systems like Mac and Linux, but may not be one-to-one. Everything else in this tutorial should be universal.

1. The first thing we need to do is install git, which you can do here for Windows: https://gitforwindows.org/ or here for Mac and Linux: https://git-scm.com/downloads

2. Click download, and use all of the defaults, **except** for when it asks you which default editor you would like Git to use. I would recommend changing this to something other than Vim (I use notepad++).

3. After you finish installing, select the box that will launch Git. This will open up a **shell** (synonymous with console/terminal) that looks something like this:
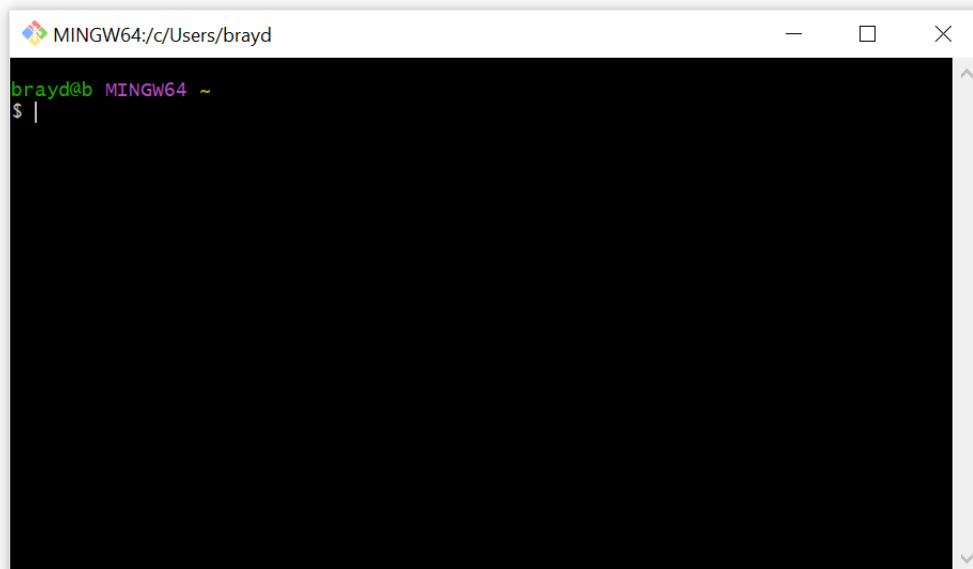
3 Initializing Git

We want to start by initializing Git by setting up a username and an email. Do that by running these two commands within the shell (include the quotations, do not include the brackets).

```
git config --global user.name "[your name]"
git config --global user.email "[your email]@[domain].com"
```

Note: This username and email address have **nothing** to do with your Github username and email! They can be changed at any time by running the appropriate config command. All this does is allow Git to label any changes that are made, so that you know who is making what changes.

Now that we have Git initialized, we want to connect it to a repository (or **repo**) in Github.

4 Making a Repo

A **repo** is just a place to store your files. A **local repo** can be thought of as just a folder on your computer. A **remote repo** is usually cloud-based. Think of a Word Document saved on your computer (local) vs a Google Doc saved on your Gmail account (remote). Github is a place to store remote repositories and connect them to local repositories on your computer.

If you don't have a Github account, go to github.com and click on **sign up** in the top right corner. I recommend that you do not use your student email since you may lose access to it in the future.

Once you've created an account,

1. click on the icon in the top right corner and click on **Your repositories**. It should look something like this:
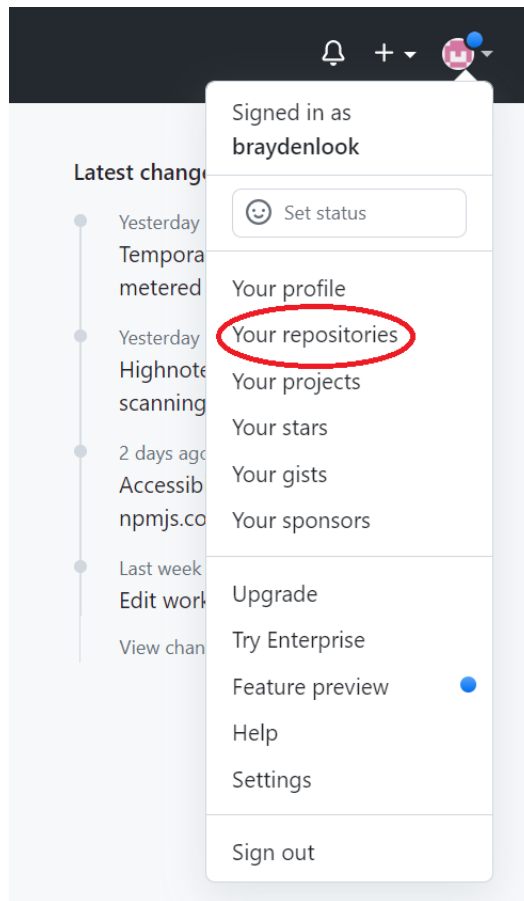
2. Enter a name for your repository. In this case, I chose "test_repo". Typically we would select the box that adds a README file. For this tutorial, leave that box unchecked.

3. Click on the green **Create Repository** button on the bottom right of the screen. You should end up with a page that looks like this:

4. Copy the address given in the textbox (make sure HTTPS is selected). This will be necessary for the next section.

5   Committing, Pushing, and Pulling

Now that we have a remote repo, we need to **clone** the repo to a local directory. All this means is copying all of the files in a remote repository to a local folder on your computer.

1. In a directory you're comfortable with, create a new folder titled "project_folder". I placed mine in the default path, i.e., "C:\Users\brayd\project_folder"

2. Open up the Git Shell

3. Check what folder you're in by typing "pwd"

4. Now navigate to the folder you just made. If you're already above the folder you need to be in, you can just type "cd project_folder"

5. Now we would like to clone this repo into our project folder. Do this by typing "git clone [HTTPS link you copied in Section 4]". Here's what it looks like for me:

When you do this for the first time, you may be prompted with a login window. This is because Git needs permission from Github to connect to the repo. To grant permission, you need to generate a **token** for Git. You may also get a warning that says your repo is empty.

6. To generate a token, go to https://github.com/settings/tokens and click on "Generate new token" in the top right.

7. Add a note if you'd like, and in the section that says **Select scopes**, select the **repo** box on the top.

8. Copy the token.

9. In the login window that opened when you tried to clone, use the token to grant access.

Now let's navigate to that folder in the file explorer:
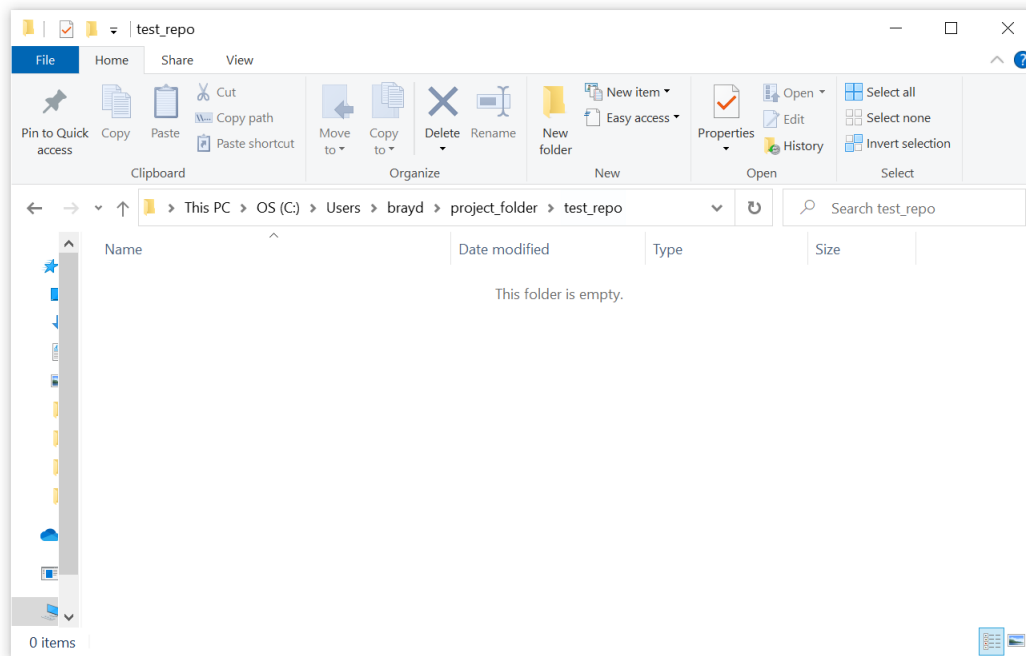
Note that it appears empty (which should make sense, since we don't have any files in our repo). However, if we click on the **View** button at the top and then select **Hidden items** in the Show/Hide box, we'll see a folder named .git.
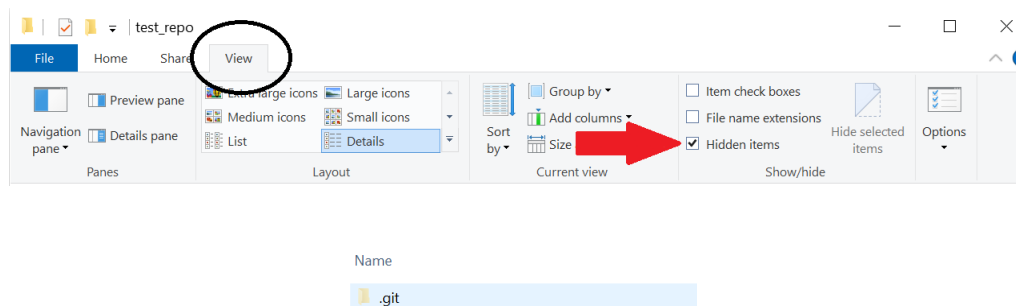
This tells us that this is being tracked by Git. Without this folder, any commands used in Git will not work properly.

Now let's add some files. Recall that using Git just means using a collection of commands to do version control; that means that we can do it in a variety of ways.

Let's start by using Github directly:

1. Start by clicking on **creating a new file** link on the repo page.

2. In the textbox on top, name the file "README.md" and in the main textbox for the file, write **This is a test file made from Github.** The .md extension allows certain formatting tricks, which we will cover later.
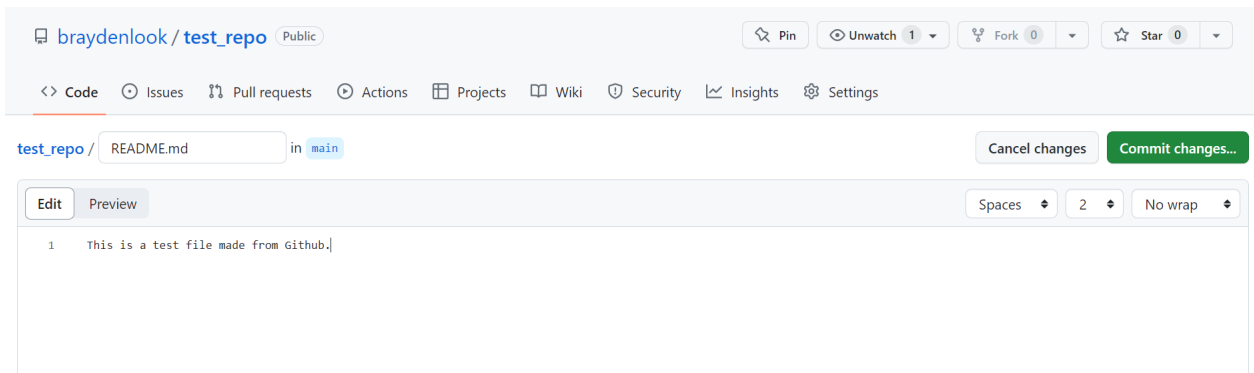
3. Then click **Commit changes...**. In general it's good practice to leave a brief commit message with searchable key-words, and then a longer explanation if necessary in the **Extended description** box. More detail is usually better than less!

   Now our remote repo has a new file, but our local repo does not. That means that we need to **pull** the new changes made to the repo.

4. Open up the Git shell again, and type out **git pull**.

5. Now check your project folder. There should be a README.md file there!

   Now let's trying committing and pushing from Git:

6. Open the README file and add a sentence that says **This was modified locally.** Save the file and close it.

7. Open the Git shell again and type **git status**. There should be a message like this:

This message tells you that you have made changes to your local project that differ from the remote repo.

8. To commit and push, start with **adding** the file by typing "git add README.md"

9. Now commit the changes and add a message by typing "git commit -m 'local commit from the shell'"

   Note that committing from Git does not change the remote repository, it only tracks the changes you've made locally. Only when you **push** the changes will they be reflected in the remote repo.

10. Push your changes by typing "git push".

11. Now check the github repo. You should see something like this!
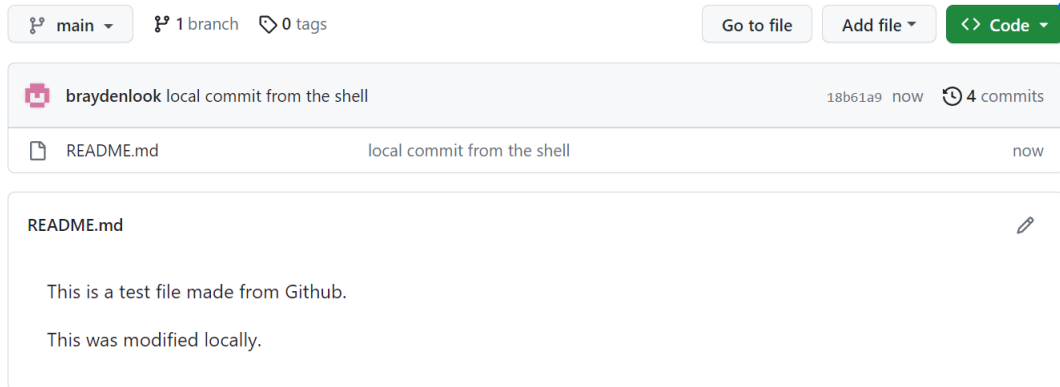
6 SourceTree

Using Git from the command line offers more flexibility, but is easier to make a mistake, since you're somewhat blind.

Making changes through Github directly will reduce the number of mistakes you make, but can be cumbersome to use. If you're on a project with multiple people, I recommend using something like SourceTree. SourceTree is to Git as RStudio is to R. Most things you can do in Git you can do in SourceTree, but with a much more intuitive UI.

1. Download SourceTree here: https://www.sourcetreeapp.com/

2. In the installation process, click "skip" when prompted to connect a bitbucket account

3. Eventually a window will ask you if you want to install Git or Mercurial (another version control program). SourceTree *should* recognize that Git is already installed on your computer. Deselect Mercurial.

With SourceTree, you can either use an existing local repo, or you can create a new local repo by cloning a Github repo (like we did manually with Git). Let's start by using our existing local repo:

1. Create a new tab by clicking the "+" symbol near the top of the application.

2. On the menu bar, click on the folder symbol that says **Add**.

3. Where it says **Working Copy Path**, click on **Browse**, and choose the folder where your local repo is (the "temp_repo" folder in our case).

4. Click "Add".

5. That's it!

If we want to clone a repo directly from Github:

1. Create a new tab by clicking the "+" symbol near the top of the application.

2. On the menu bar, click the **Clone** option.

3. Where it says **Source Path / URL**, paste the HTTPS code that we used earlier to clone with Git.

4. In **Destination Path**, pick the name of the folder where you'd like to store your files.

5. Click **Clone**.

Let's modify a file to see why SourceTree can be so useful.

1. Start by opening up your README.md file, and add a line that says "This is a test change for SourceTree."

2. Open up SourceTree. After a moment, there should be a number 1 icon next to the "Commit" button on the top left. This is indicating that you have one file that has changes. Click on it.

You should see something like this:
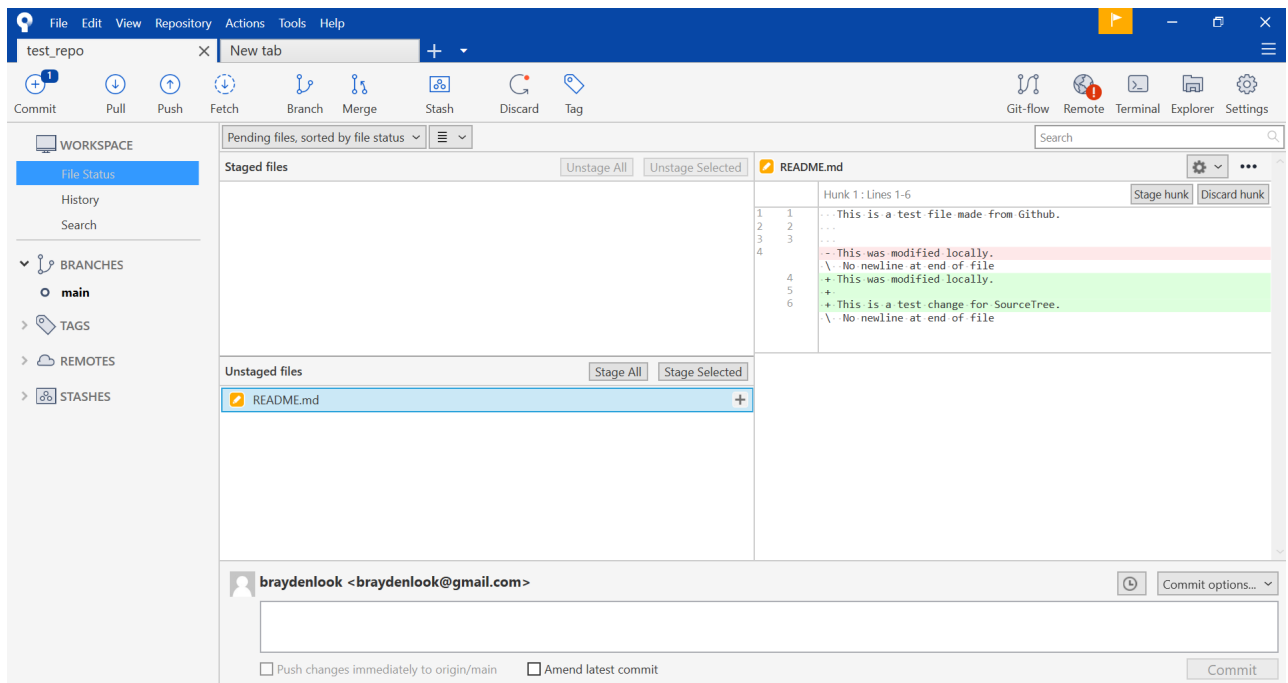
On the bottom-left side of the screen are your **Unstaged files**. These are the files that have detected changes. On the middle-right side of the screen are the changes to the file. The red highlight indicates lines that have been removed, and the green highlight indicates lines that have been added. On the middle-left is the staging area.

1. Click on the README file in the **Unstaged files** to highlight it if it isn't already highlighted.

2. Click **Stage Selected**.

3. On the bottom of the page is a box for leaving a commit message. Type "This commit was done via SourceTree."

4. Click **Commit** on the bottom right of the screen.

You should be put back on the **History** page, and now you can see the change you made in the timeline! If you click on the commit in the timeline, you can see the files that were changed and the specific lines that were added and removed.

You'll also see the number 1 next to the **Push** button on top. For the changes to be made in the remote repo, click on **Push**, and then when prompted with a new window, click **Push** again in the bottom right.

It might not be that obvious how convenient SourceTree can be from this demo, but now imagine you have a folder with 20 different files, each with hundreds of lines of code that are being changed by different people. In a few clicks you're able to see who made changes, what files were changed, what lines were changed, and when the changes were made.