

GitHub Activity

Objective: To install GitHub on your computer/laptop and understand how it works. In this activity, you will be creating a repository on GitHub and then adding a dummy project to it. We will be adding and manipulating text files in this lab just to see how Git works, but this can be extended over to projects and such.

Part 1: Install Git

First, you have to install Git using the provided link in the activity module. Note that the installation process differs from OS to OS. On Mac/Linux, installation is fairly straightforward, but Windows is a bit different. Here, I will be demonstrating how to install and use Git in Windows.

Please use this link to install Git on your system.

<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

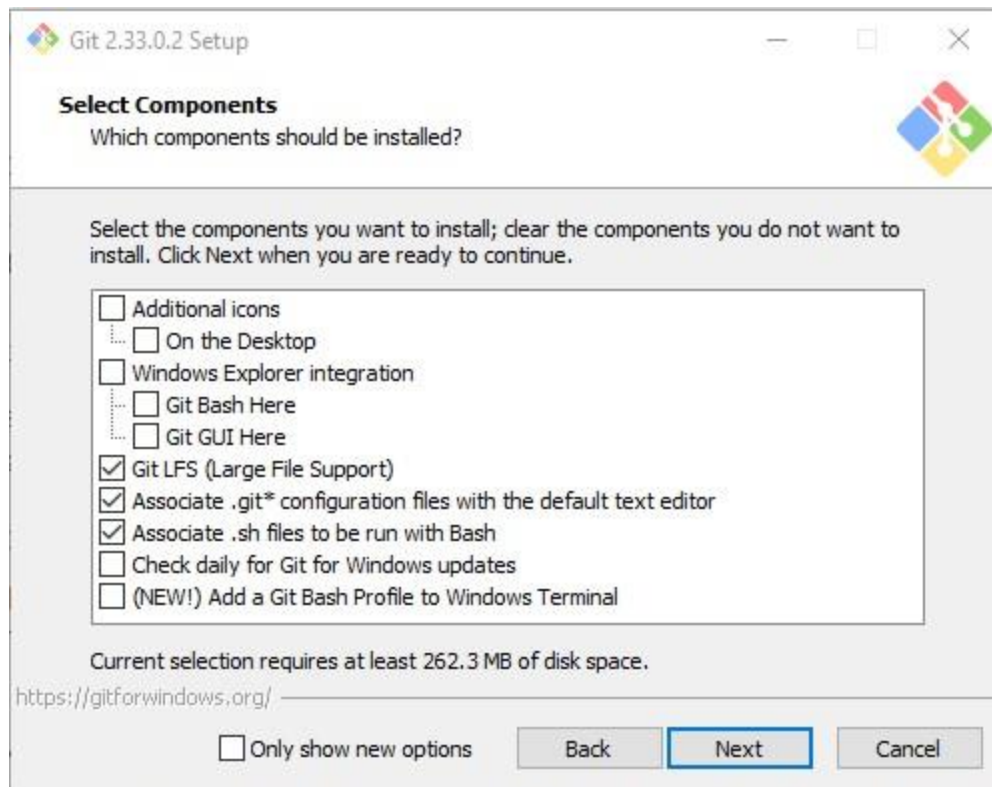
Installing on Windows

There are also a few ways to install Git on Windows. The most official build is available for download on the Git website. Just go to <https://git-scm.com/download/win> and the download will start automatically. Note that this is a project called Git for Windows, which is separate from Git itself, for more information on it, go to <https://gitforwindows.org>.

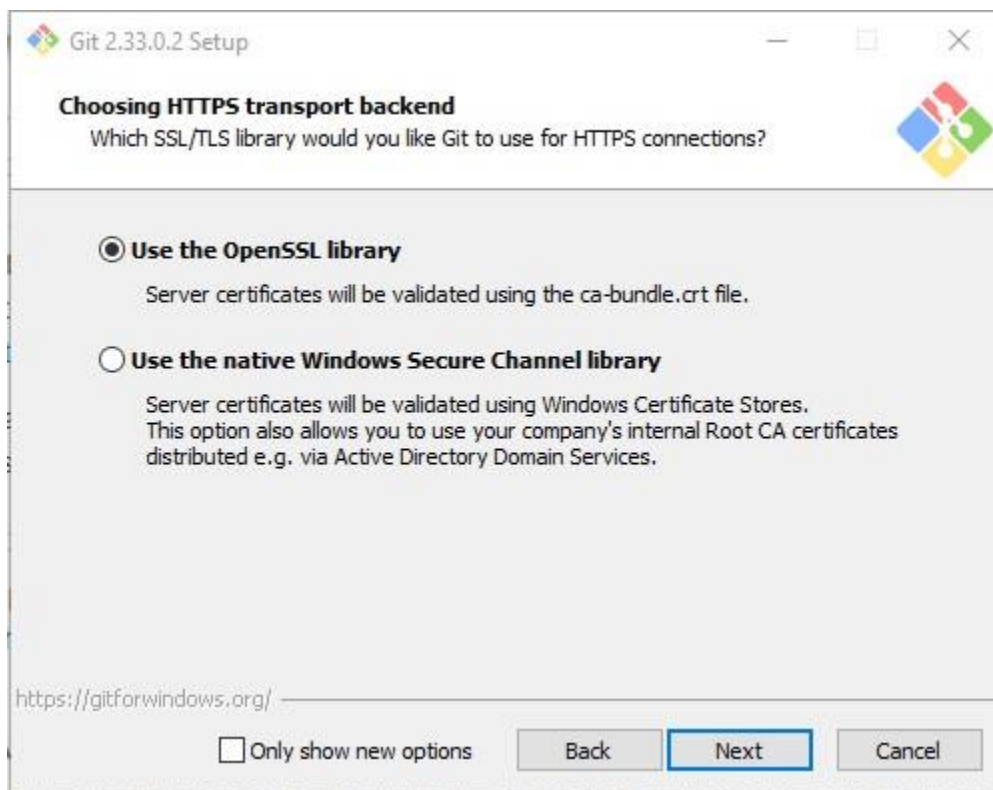
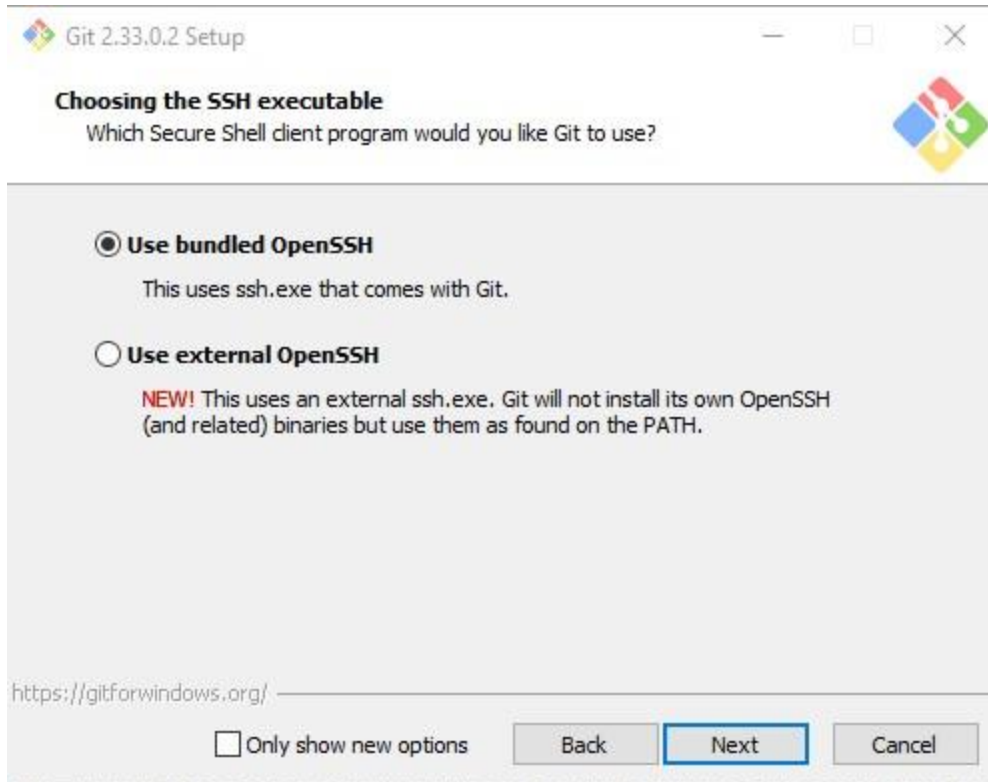
To get an automated installation you can use the [Git Chocolatey package](#). Note that the Chocolatey package is community maintained.

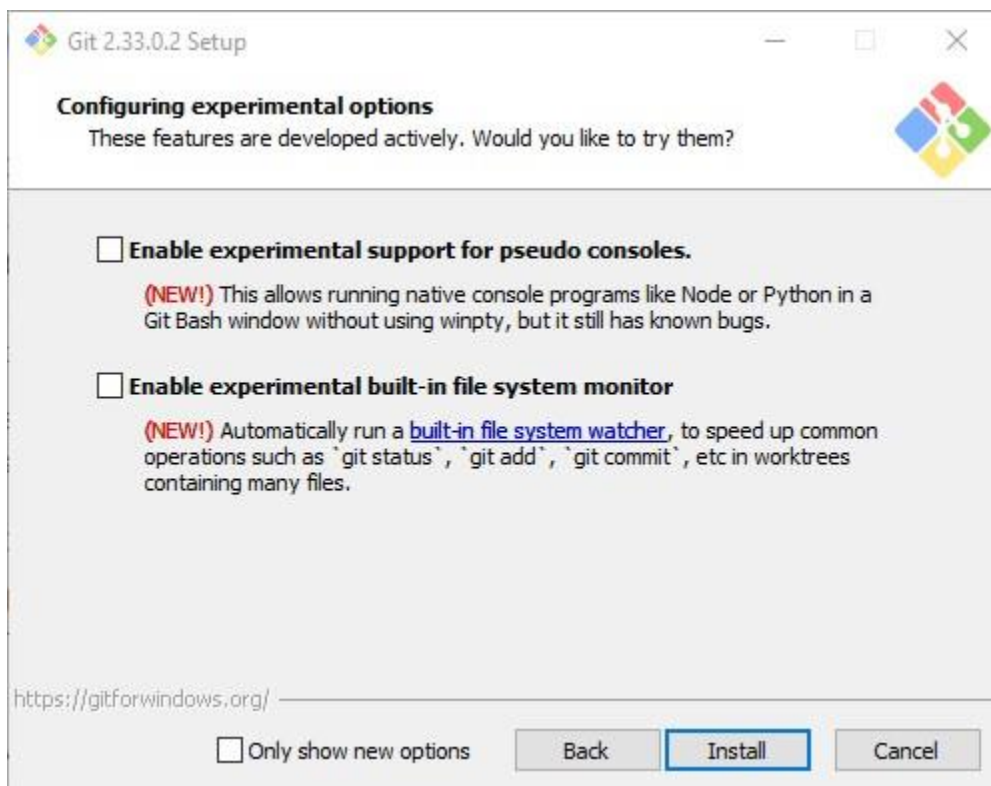
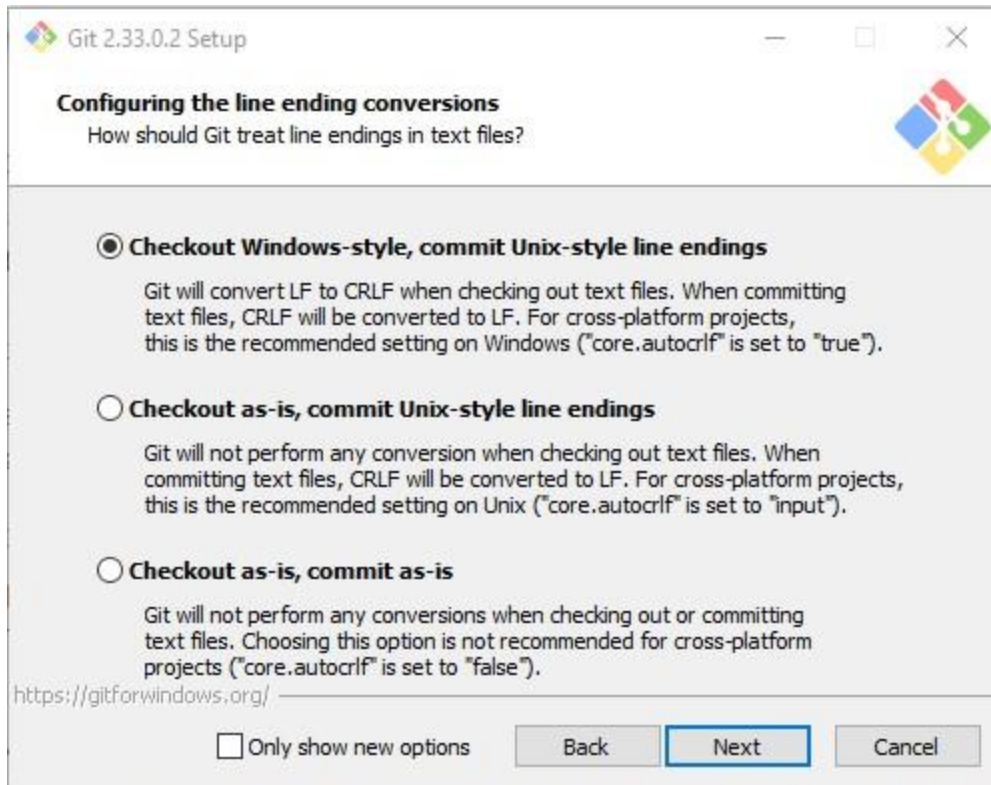
Install Git by clicking on the highlighted link. This will automatically start the download process. Once it is finished downloading, start the executable file that is downloaded.

For the installation process, you can select the default components that Git will install for you. Below will be screenshots for more ambiguous areas.



GitBash is not needed for our purposes but if you are more comfortable using a Unix like environment, then you may install GitBash. Likewise, Git GUI is not needed as well.





These are all the configurations I used to install Git on my computer. To verify that the installation process was completed successfully, open Command Prompt or Terminal and type in either “git” or “git --version”

```
CA. Command Prompt
init                Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add               Add file contents to the index
  mv                Move or rename a file, a directory, or a symlink
  restore           Restore working tree files
  rm                Remove files from the working tree and from the index
  sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
  bisect            Use binary search to find the commit that introduced a bug
  diff              Show changes between commits, commit and working tree, etc
  grep              Print lines matching a pattern
  log               Show commit logs
  show              Show various types of objects
  status            Show the working tree status

grow, mark and tweak your common history
  branch            List, create, or delete branches
  commit            Record changes to the repository
  merge             Join two or more development histories together
  rebase            Reapply commits on top of another base tip
  reset             Reset current HEAD to the specified state
  switch            Switch branches
  tag               Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
  fetch             Download objects and refs from another repository
  pull              Fetch from and integrate with another repository or a local branch
  push              Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.

C:\Users\Eric>
```

```
CA. Command Prompt

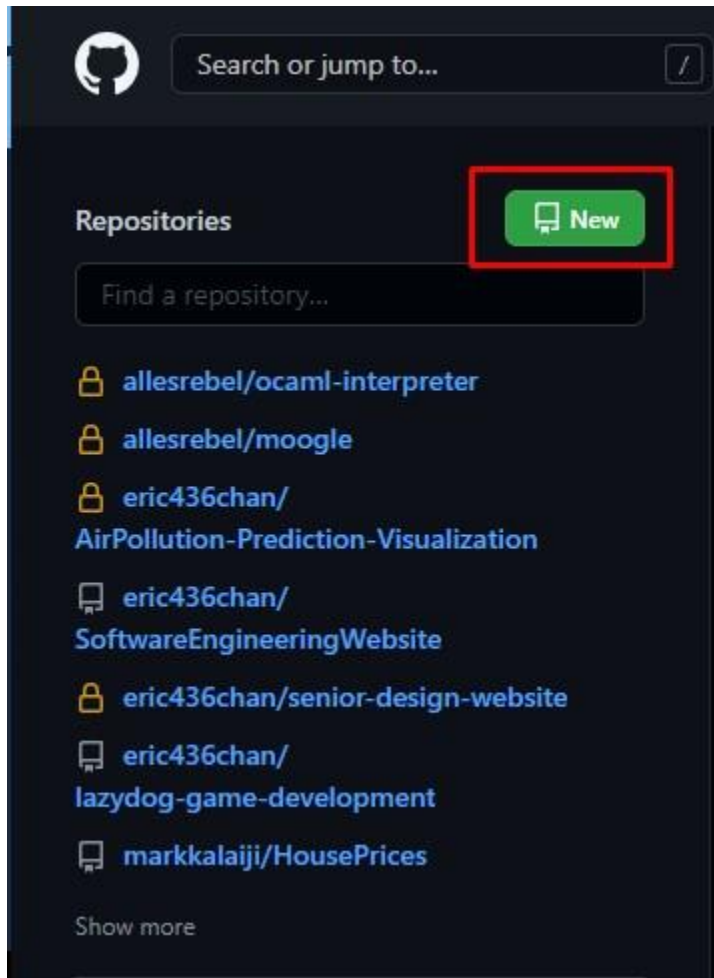
C:\Users\Eric>git --version
git version 2.33.0.windows.2

C:\Users\Eric>
```

You should see something similar to what is being displayed in these images. If you don't, please contact one of the mentors and we'll assist you in install Git.


Part 2: Creating a Repository



If you don't already have a GitHub account, you'll need to create one first. Once you have logged in, you'll need to create a repository. There will be a button on the left that says "new". Click that button to create a repository.



Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Owner *  **Repository name ***

 / 

Great repository names are short and memorable. Need inspiration? How about [musical-enigma?](#)

Description (optional)

☐  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☒  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

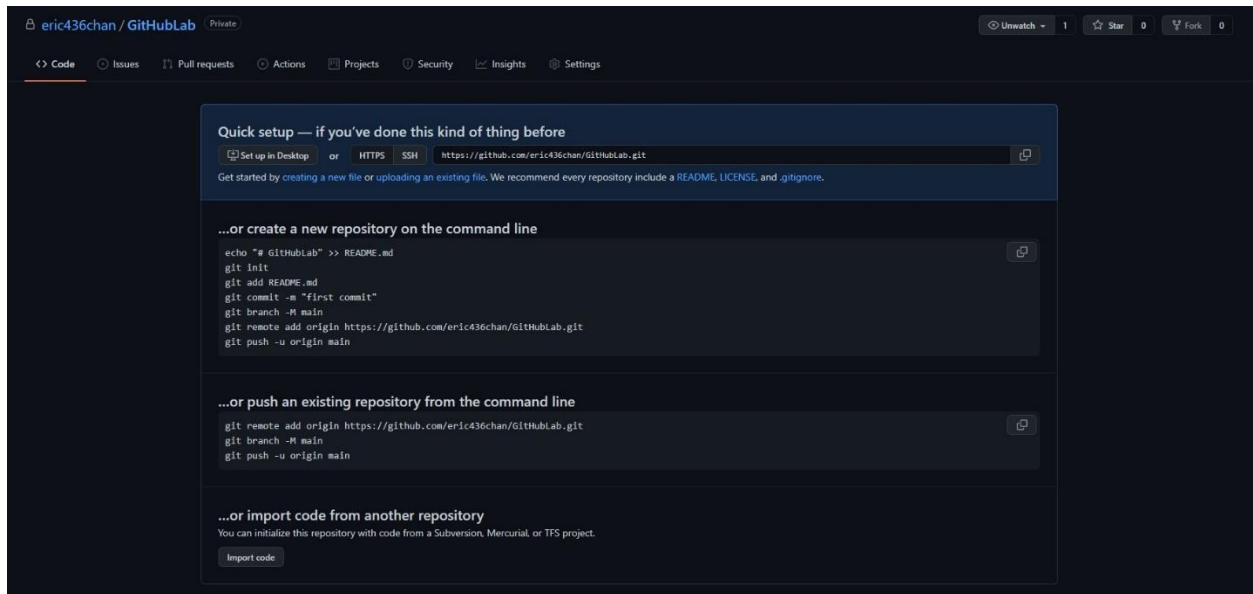
☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

The name can be whatever you want it to be, however, I recommend you name the repository something meaningful and not something random. Second, make sure the repository is private. Afterwards, click the “create repository” button.

You should see something like what you see below after creating your repository. You'll see that GitHub already has a list of commands to help you connect a project to the repository. **Keep in mind the HTTPS link to the repository, you'll need it later.**



Part 3: Changing the Repository

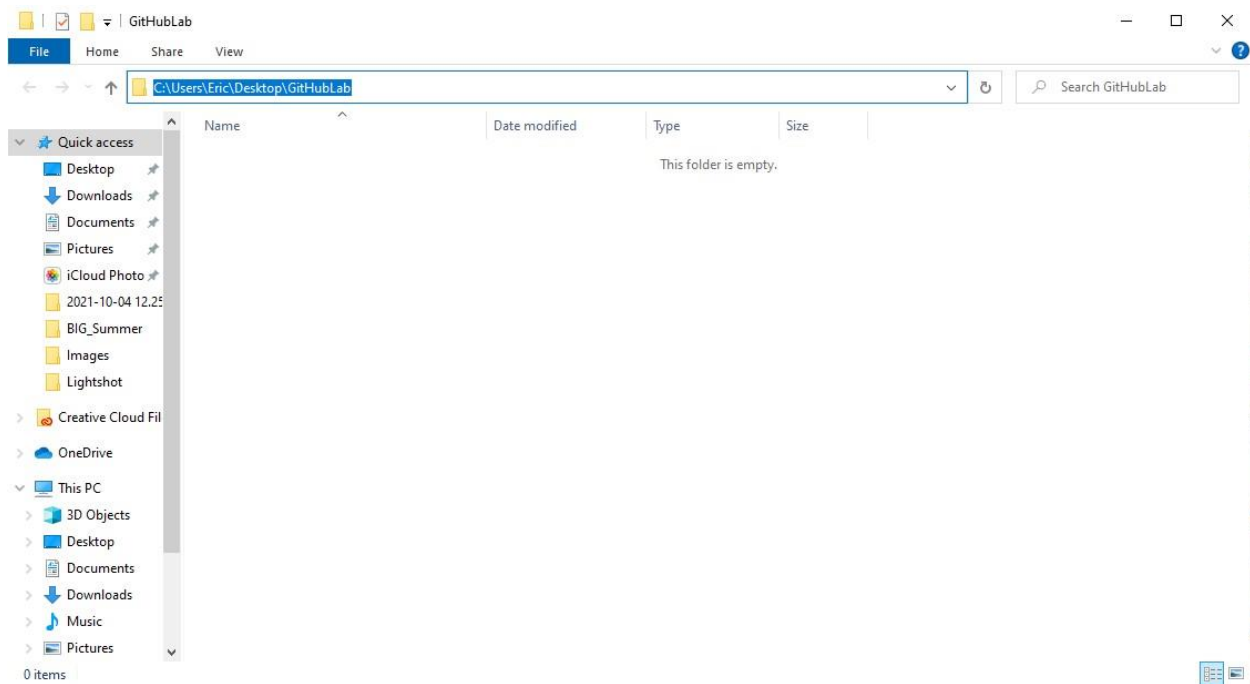
We'll now get started with the main section of the lab which is using Git to make changes to a repository. First, we need to connect a project to the repository. Usually, this is just the source folder. For our purposes, we'll create a dummy folder.

Create a New Folder on your Desktop or in a secure location. **Keep in mind the location where you created this folder.** You can name the folder whatever you want in this lab, but again, make sure the name is meaningful.

After creating the folder, create a text file inside the folder and fill it with some filler text.

Now, you're ready to make some changes to your repository. Open up Command Prompt or Terminal and navigate to where you created this folder.

This can be done by clicking the magnifying glass on the toolbar and typing in **cmd** if you're on Windows and **terminal** if you're using Mac. Once you have that opened, open the folder and copy the path.



Once copied, type this command into the Command Prompt or Terminal and hit enter.

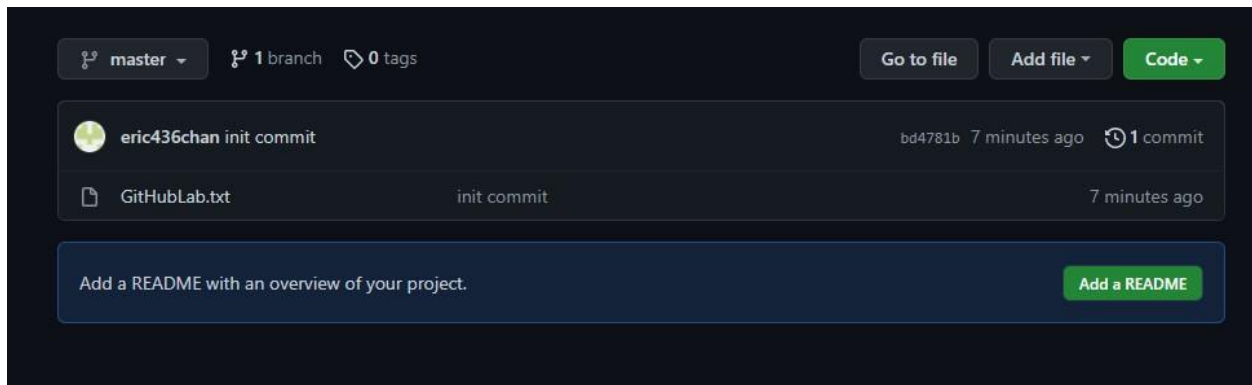
cd <path>

From there you will type these commands in order.

1. **git init** (This initializes a GitHub repository at this folder)
2. **git add .** (This adds all current files to the repository)

3. **git commit -m “init commit”** (This makes a commit action to the repository. The -m signals a message that you want attached to this commit. It is good practice to have a meaningful message attached to your commits)
4. **git branch -M <name of branch>** (This creates a new branch for your repository. -M signals that this will be the main branch. General naming convention for the main branch is usually “master”, “main”, “prod”, etc. As long as it has a meaningful name, it will be fine.)
5. **git remote add origin <link to repository>**
6. **git push -u origin <name of branch>** (This pushes the current commit up to the current branch. -u means upstream or in other words, we want this branch to track any changes to the specified branch. Make sure the name of the branch here matches the previous one)

Afterwards, you should see something like this.



Next we will be creating new branches. Branches are a great way to ensure that we are not messing up anything we have current. We generally want to branch out whenever we have something new to add.

In the Command Prompt or Terminal, assuming you still have it open, if not, navigate back to the folder you created. Type these commands in order.

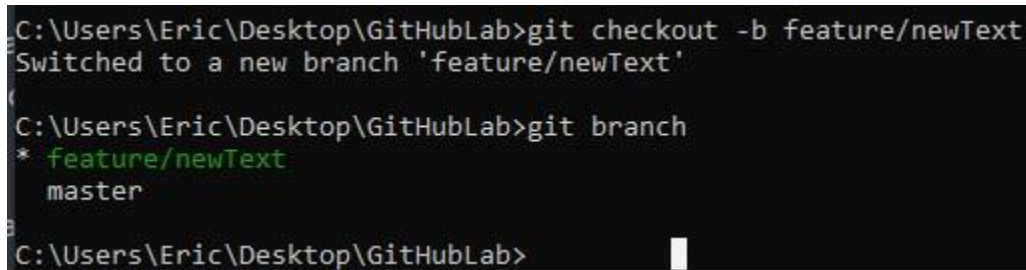
1. **git branch <branch name>** (While you can name your branch whatever you want, general convention goes something like this: feature/<whatever feature is being added>)
2. **git checkout <branch name>**

Note that the above commands also have an equivalent shorthand

git checkout -b <branch name>

You can also do a sanity check by typing in **git branch** and it'll list all the current branches we have as well as the current branch we're on.

If done correctly, it should look something like this.



```
C:\Users\Eric\Desktop\GitHubLab>git checkout -b feature/newText
Switched to a new branch 'feature/newText'

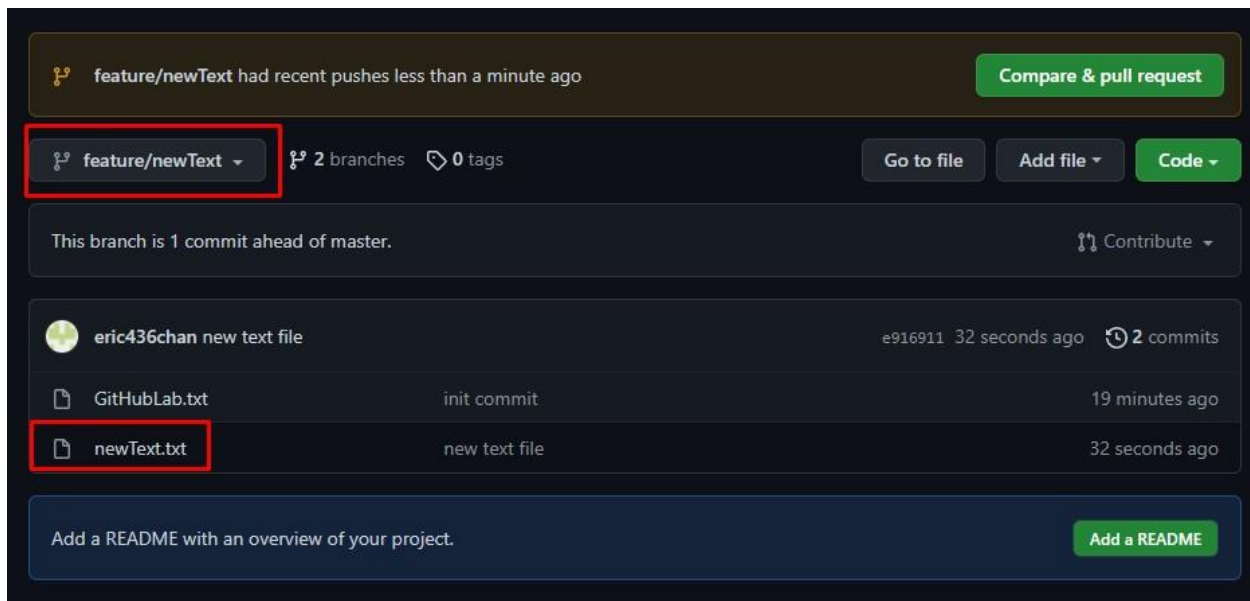
C:\Users\Eric\Desktop\GitHubLab>git branch
* feature/newText
  master

C:\Users\Eric\Desktop\GitHubLab>
```

Now, just because we've made a new branch, does not mean it'll show up on GitHub. We actually have to make a commit first before it appears. So like before, create another text file and add some filler text in there. Enter these commands in order.

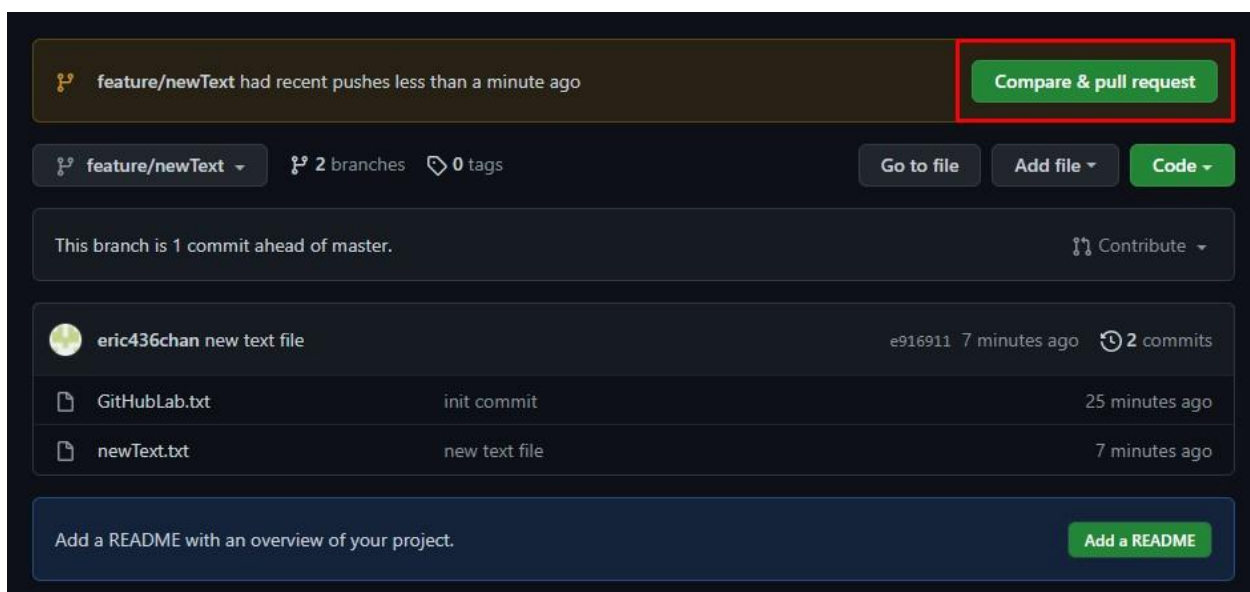
1. **git add .**
2. **git commit -m "new text file"**
3. **git push -u origin <branch name>** (for me, this will be feature/newText)

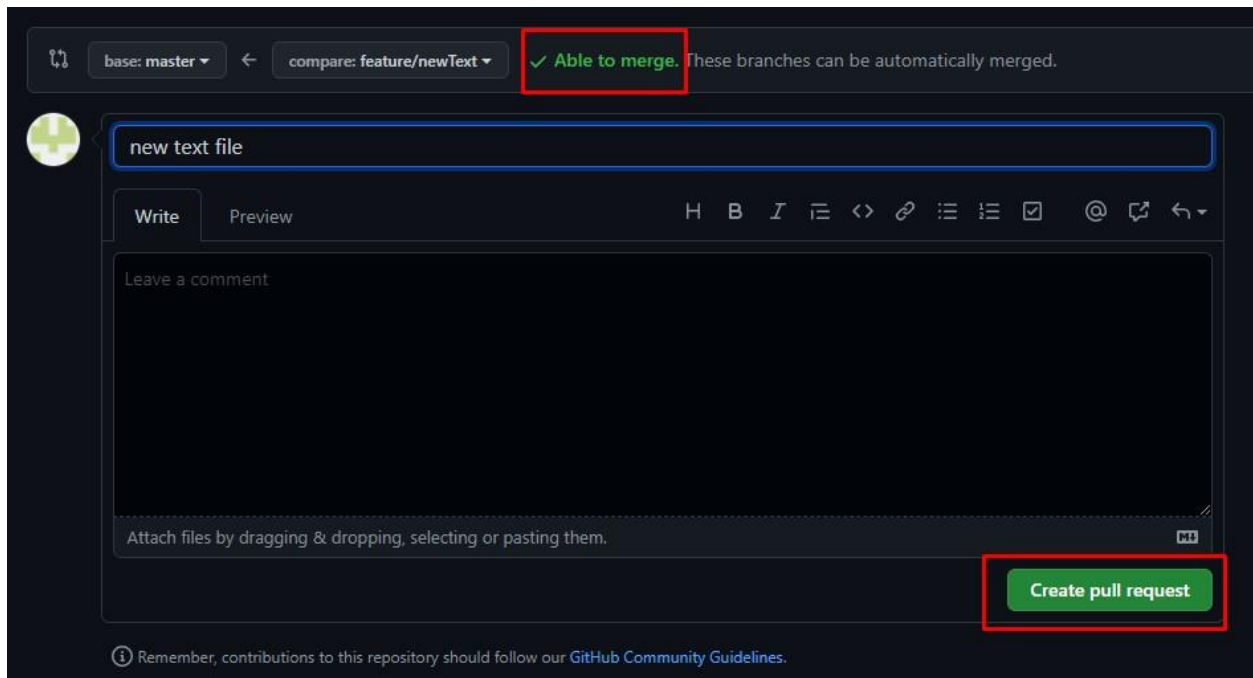
If we check our GitHub repository, we can see our new branch as well as what we added which is a new text file. Note that this new branch has this new text file but the main branch does not. This allows us to make adjustments while not messing with code that is already established.



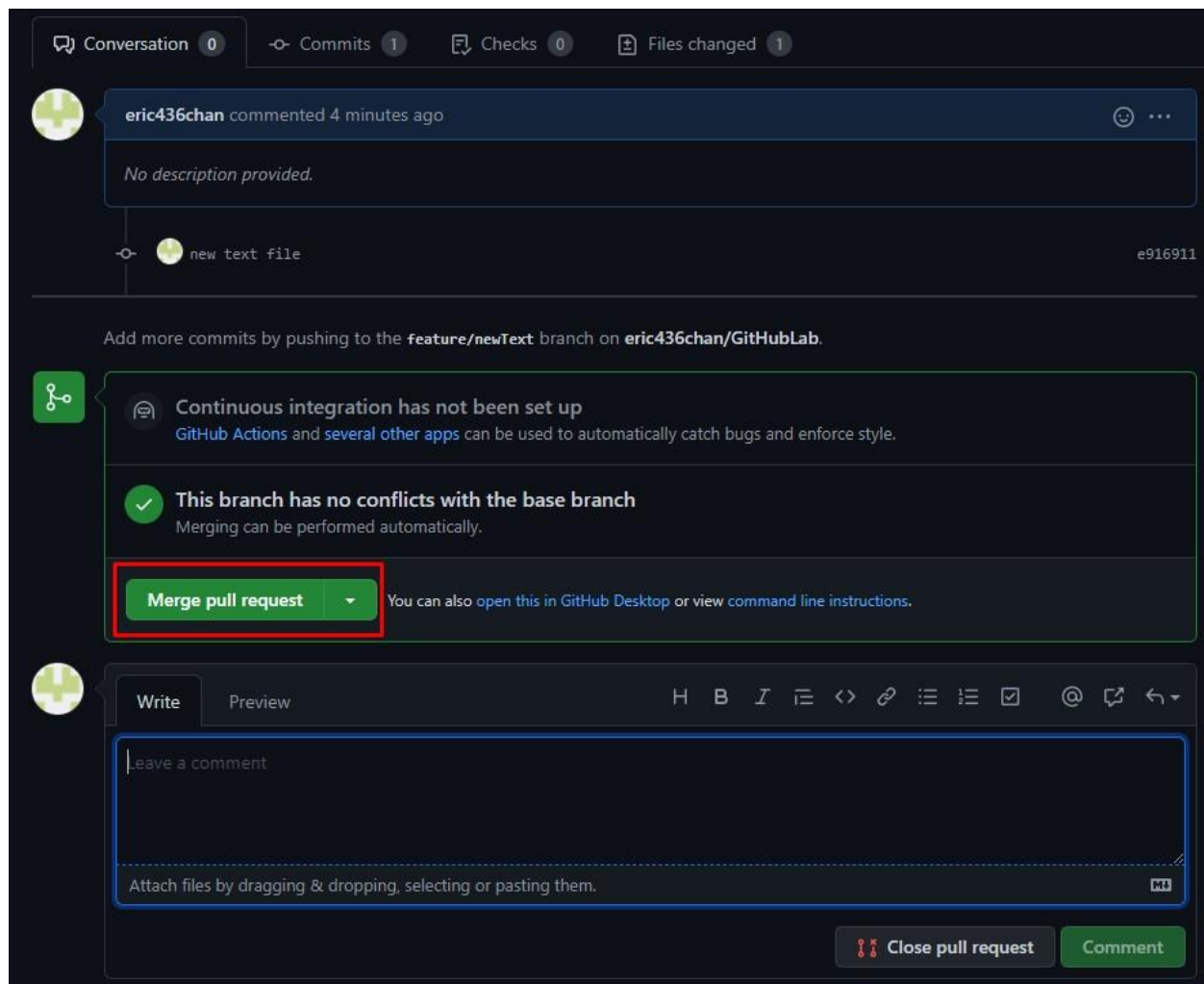
Part 4: Merging

Merging is simply combining two different branches into one. Note that this can have some issues depending on what you have in the two different branches such as if the two branches **BOTH** made adjustments to the same file in the same location, it can mess things up or not merge entirely. This is why it's important to make sure you know who is working on what and not to touch anything outside of what you are working on to ensure everything works properly.





Note that it says “Able to merge” here. If there are errors, you can still merge, though it is not recommended. You’ll have to fix any errors manually once you merge.



After you merge, GitHub will ask you if you want to delete the branch. While you are not required to, it is more optimal to delete the branch. However, for our case, keep the branch, as we will be using it for additional purposes.

You'll see now on your main branch that it contains both text files.

Note that the merge command can also be done in the Command Prompt or Terminal but I always find this way to be easier.

Part 5: Pulling in changes

Pulling is simply, pulling any new changes to our current repository. Usually, you'll be pulling from the master branch because that branch would contain changes made by other members of your team. This is probably the most critical bit of Git (at least amongst the basics). To avoid any merge conflicts, try to pull from the master branch before working and before pushing.

First return back to your main branch using this command

git checkout <main branch name>

Check in your folder. You'll notice that the newText.txt file isn't there even though it's in the repository. That's because we haven't pulled in the changes yet. Now type in this command.

git pull

You should see the text file now. Note that this only works because we've set our upstream. Normally you'd have to specify which branch to pull from but because we've set an upstream, we can bypass that part.

Part 6: Deleting branches

Once you've actually finished working on a branch, normally you'd delete it. This would have been done when we merge but since we didn't, we can go ahead and delete it now.

Type in this command

git branch -d <branch name> (Hopefully you didn't decide to delete your main branch and you've decided to delete your feature branch)

To check whether the branch was successfully deleted, use this command to check

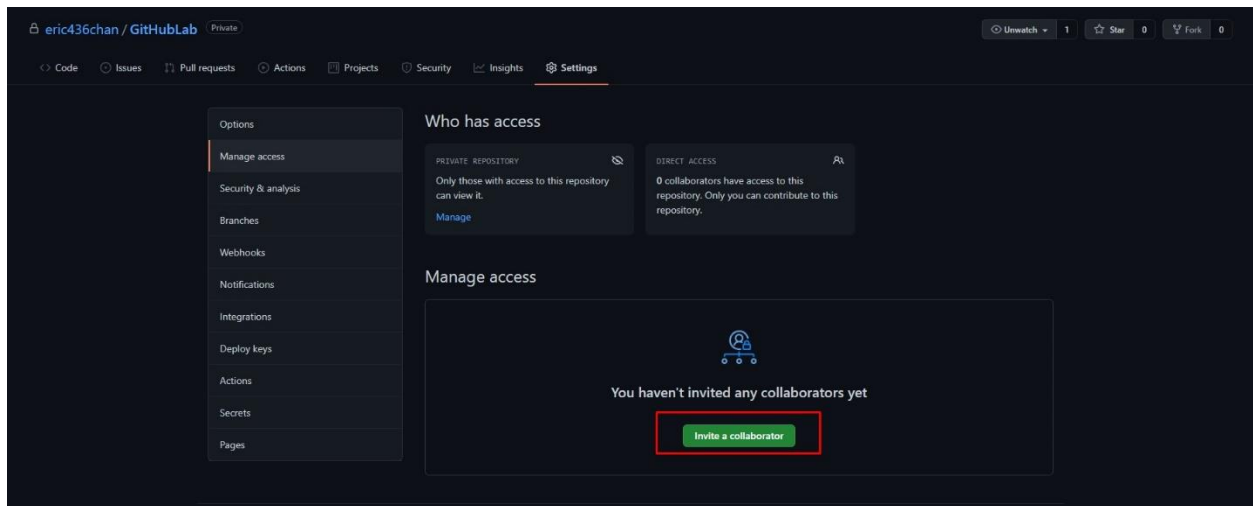
git branch

You'll notice the branch was successfully deleted. However, this only deletes the local branch or the branch that's stored on your system. The branch still exists in the repository. To get rid of the remote branch you'll have to use this command

git push -d origin <branch name>

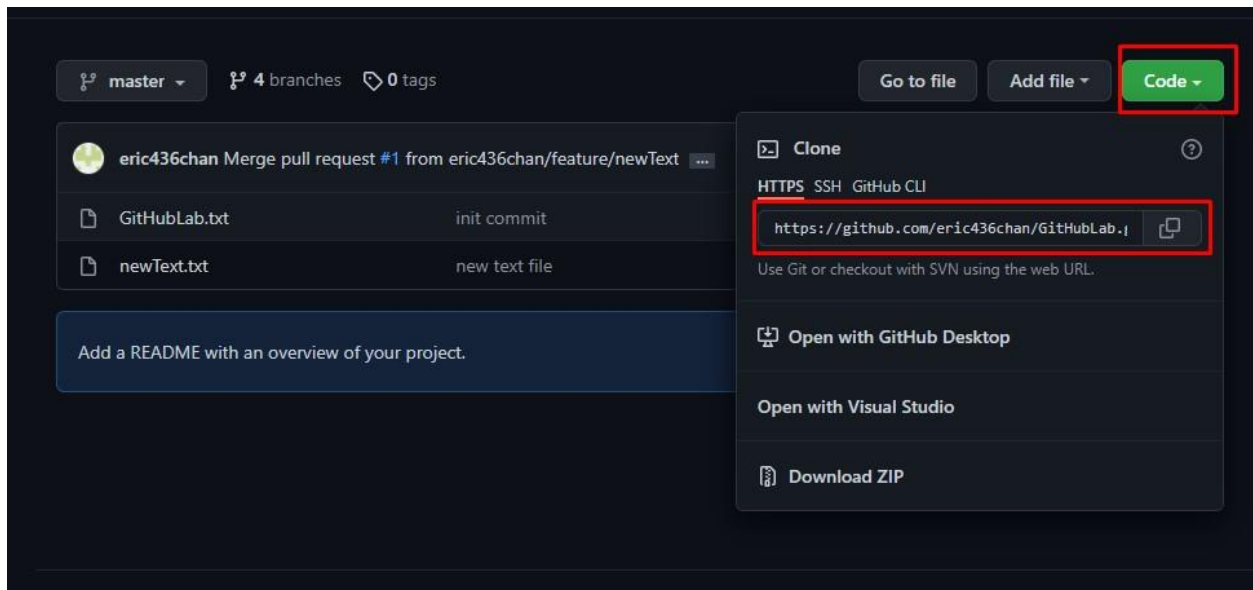
Part 6: Adding people to your repository

Adding people to your repository is probably the easier part of using Git.



Click on settings and then manage access. From there you should be able to invite collaborators to your repository. Collaborators are allowed to make new branches and commits as well.

Adding collaborators to your repository is not enough to have them start working on the project. They need to have an actual connection to the repository. For this, they'll have to clone the repository.



When you're cloning, make sure you're on the main branch of the repository. Afterwards, copy the https link of the repository and then open Command Prompt or terminal and type in this command.

git clone <link>

This will clone the repository to wherever your current directory is. Now you're ready to start working on the project.

Additional

If you'd like to add the previous Android Studio Lab to your GitHub, you simply need to make a new repository, navigate to the folder that contains all the work for the lab and follow the first few commands from **Part 3**. It does not have to be a new project in order to initialize git.

Conclusion

In this lab, you have successfully learned how to create a repository and how to use the commands to make changes to the repository. Of course this is Git on a basic level. If you want to learn more, take a look at the provided resources for more information.

