# Configuring GitHub



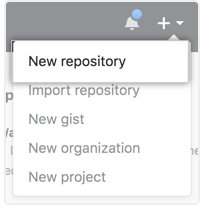
At this point you should have two files added for tracking and committed for a push to a remote repository. These files should be both a `.gitignore` and a `README.md` which was to be completed as part of last week's exercise. For us to get these projects up on GitHub, we’ll need a repository for the project to live in.

We’re going to start this week’s lab off by creating a remote repository on GitHub. After we’ve created the repository, we’ll execute a couple Git commands in our local project folder. After we’ve connected our remote repository to our local project, we’ll generate some SSH keys that we touched on last week and configure GitHub to use our public RSA key.

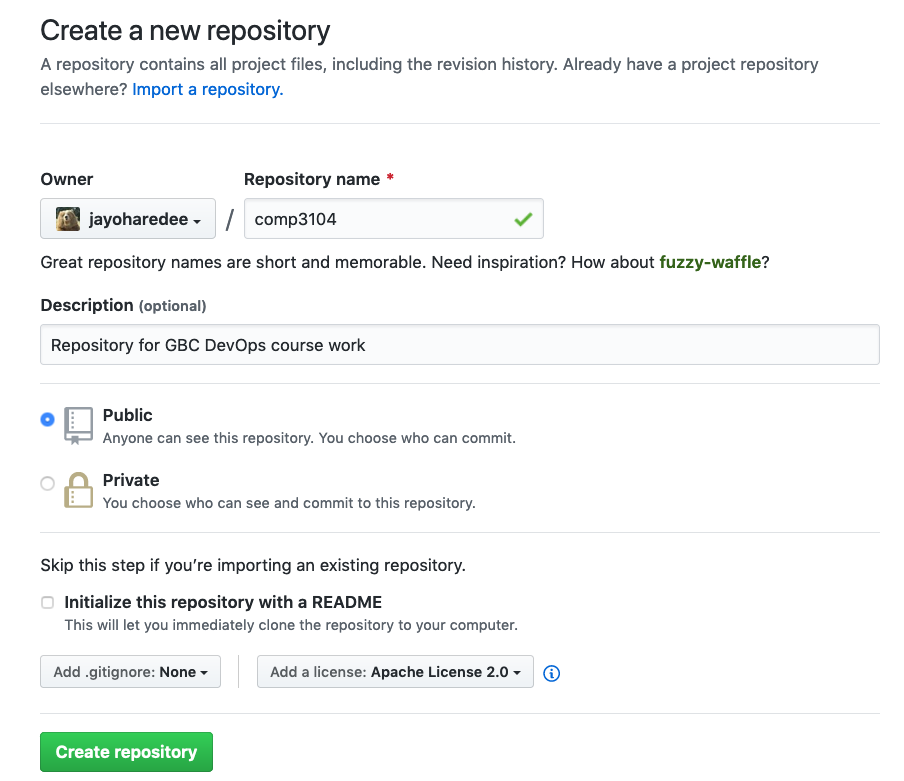
## Creating a remote repository with GitHub

If you have not yet set up an account with [GitHub](https://github.com/), now would be the time to do so. You can sign up with any email you please although I strongly encourage the use of your GBC address as you’ll be able to reap the benefits from their [education pack](https://education.github.com/students).

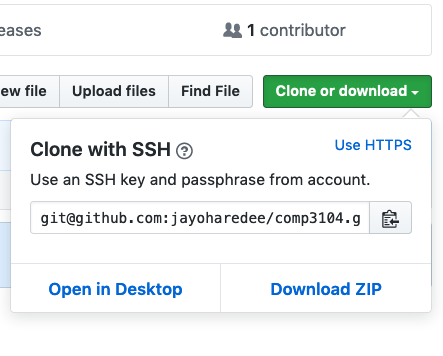
After you’ve completed the sign-up process and are logged in, you should see a + symbol in the top right corner. Click the + and in the dropdown you’ll find a ‘New Repository’ link that looks something like this:



Once the ‘New Repository’ link is selected, we should see some input fields for us to provide a repository name and description. Feel free to provide whatever description you like but for the name, please enter ‘comp3104’. There will also be the option of selecting a license, choose the Apache License just as I have in the image below.

After this has been completed, please click the ’Create repository’ button.

Once the repository has been created, we will be presented with a new screen showing our license file. On top of our newly generated license file there should be another green button for us to click that says, ‘Clone or download’. Please click this button and when the dropdown appears ensure it has a header of ‘Clone with SSH’.



In the above image, we’ll want to copy the text that’s in the input field. We can do this by either selecting the text and copying to our clipboard through keystrokes, or we can click on the clipboard icon to the right of our repository address.

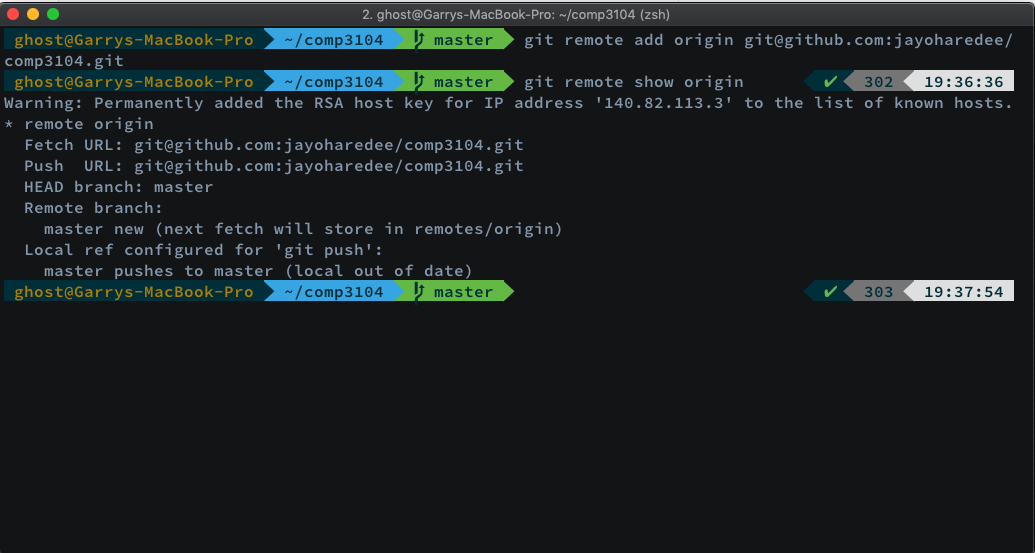
Once that’s copied, we’ll have to set this as our remote address in our local project directory where we have created our .gitignore along with our projects readme. Referencing the commands from last week’s lab class, we can see to set a remote repository address we must use the following command:

`git remote add origin COPIED\_REMOTE\_ADDRESS\_HERE`

In the above command, the all caps snake cased text must be replaced with the address copied from the GitHub dropdown that appeared when we clicked ‘Clone or download’. Using what was provided in my dropdown, the command I need to execute will look like this:

`git remote add origin git@github.com:jayoharedee/comp3104.git`

The above command will look a bit different for you. Instead of showing `git@github.com:jayoharedee` you should see your user name instead. Let’s hop over to the terminal and enter the command. See my example below for reference.



You'll notice that I entered an additional command, this being `git remote show origin`. You may also notice the use of `origin` in the first command. `origin` is commonly used to reference the source origin of where the code is coming from. If we further examine the above image, we should also see some additional output showing both a `Push` and `Fetch` url. This is the URL we copied from our GitHub dropdown in our newly created repository.

Before we go pushing and fetching any changes, we have one last step to complete. That is the setup of our SSH keys. Without generating our RSA keys and providing GitHub with our public key, we won’t be able to make any changes as we aren’t authorized to do so. Typically, when working with remote repositories some sort of authentication has to take place as we don’t want just any random Internet dweller to modify our code base. Our authentication when making Git commits from the command line will come in the form of SSH. We briefly touched upon it last week and this week we’ll be further elaborating on the creation, configuration and usage of the protocol.

Let’s get to that right now.

## Generating SSH Keys

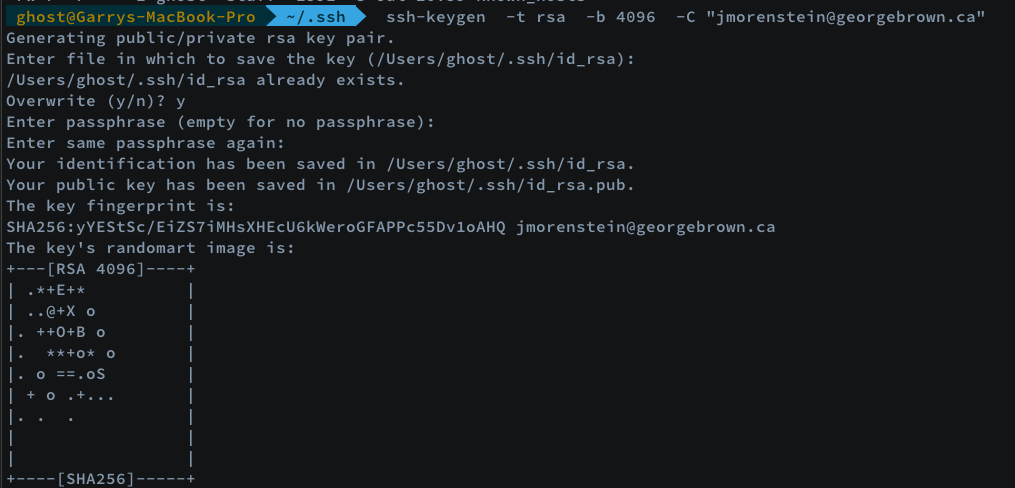
Last week we learned about asymmetric cryptography and the concept of private and public keys. A quick recap – we must keep our private keys secure and private. It’s our public key that we want to provide remote servers with in order for an authentication process to take place. We’re going to generate an SSH key pair and then take our public key and link it to our GitHub account.

To generate an SSH key pair, we’re going to assume that this has not been done. If you’ve already completed this process for another reason and would like to use the same public key, feel free to skip to the next step. For those in need of a key pair, please head back to your terminal and follow along with me.

After you have your terminal open, whether it’s GitBash for Windows or a terminal of choice in macOS or Linux, please paste the text below into your terminal but replace [your\_email@example.com](mailto:your_email@example.com) with the email you would like GitHub to use. It’s recommended you use the same email you chose to signup with your GitHub account. If you’d like to switch emails, this can be done in GitHub to correspond with your SSH email.

` ssh-keygen -t rsa -b 4096 -C "[*your\_email@example.com*"`](mailto:your_email@example.com%22%60)

We should then be prompted to “Enter a file in which to save the key”. Press enter to accept the **default location (~/.ssh).** Please see image below for my example.



When asked to enter a file to save the key, hit enter again for the default SSH key pair names (id\_rsa).

At the prompt, it will then ask you for a secure passphrase. You can enter one if you like. For me, I just trust in the security mechanisms of SSH and choose to go passphrase-less for something like this. I choose to hit enter twice instead of entering a passphrase twice. Choose what makes most sense to you.

We know have to add the key to the systems ssh-agent. The SSH agent manages our keys and remembers a passphrase if you chose to enter one. Here’s a command to do just that:

` eval "$(ssh-agent -s)"`

After entering the above command you should see something like:

> Agent pid 27867

We now must add our private SSH key to the ssh-agent. If you created your key with a different name, please ensure you replace id\_rsa with the name you chose when selecting a file to save the key. I accepted the default name for my key, so here is how my command would look:

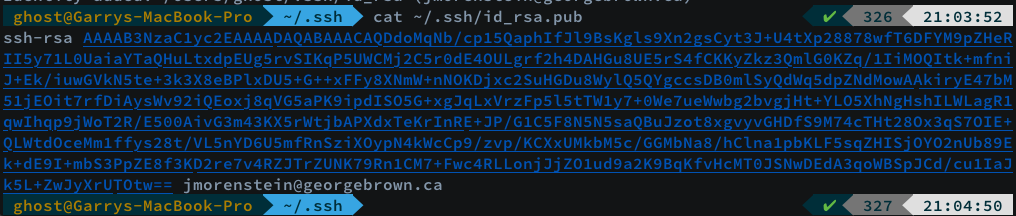
` ssh-add ~/.ssh/id\_rsa `



If all went as planned, we should be able to execute the following command to view our public key using a command we learned in our first lab, cat. Let’s give it a shot:

`cat ~/.ssh/id\_rsa.pub`

If you don’t see output like the below image, please enter your .ssh directory located in `home` and see what’s going on there. Questions, come see me.



Above is my current public key. I can expose all of this to you because you do not have access to my private key. Now that we’ve confirmed our generation of SSH key pairs, it’s time to provide GitHub with our public key so we can get to pushing and fetching code.

## Adding an SSH key to GitHub

This next step is going to be OS specific. Please find the OS your using below and follow the commands you need to get the job done. In any event, the objective is to copy our public key to the clipboard so we can paste it in GitHub. In any event, if you chose a different name for your public key, please swap out id\_rsa.pub for YOUR\_KEY.pub.

With the next step we’re going to be using a new piece of BASH syntax called the with the `<`. This key is similar to `|` in the sense that it helps us with the input and output of commands. Some refer to this as “data flow”. Using `<` after a command, we’re able to get the output of a command to flow from right-to-left. Using this command, we’re using the copy command in BASH to collect what’s in our public key file.

### Windows

In your GitBash terminal, please enter the following command

` clip < ~/.ssh/id\_rsa.pub `

You should now have your public key in the clipboard, please move on to the next step to paste the key in your GitHub settings.

## Mac

` pbcopy < ~/.ssh/id\_rsa.pub `

You should now have your public key in the clipboard, please move on to the next step to paste the key in your GitHub settings.

## Linux

If you don’t have something like xclip installed on your system and are running a Debian distro like Ubuntu or Mint, You’ll want to install it first.

` sudo apt-get install -y xclip `

After the download is complete, we’ll want to copy our public key to the clipboard:

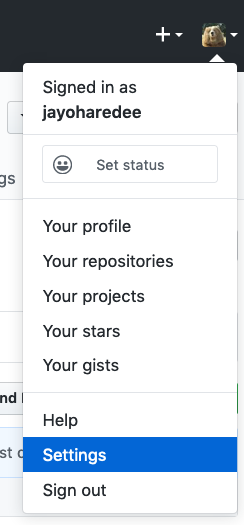
` xclip -sel clip < ~/.ssh/id\_rsa.pub `

You should now have your public key in the clipboard, please move on to the next step to paste the key in your GitHub settings.

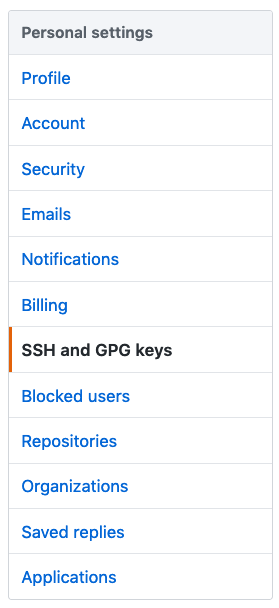
## Providing Our Private Key to GitHub

You should now have your public key in the clipboard, please move on to the next step to paste the key in your GitHub settings.

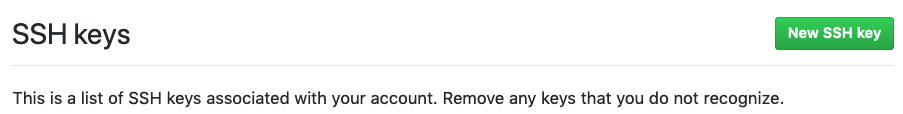
In the upper right corner of your GitHub profile, you’ll want to locate your avatar image and select the ‘Settings’ anchor from the dropdown. Mine looks like this:



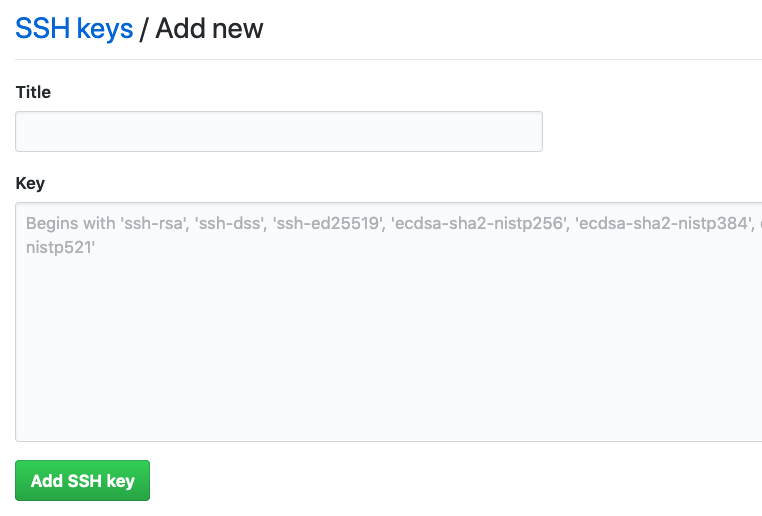
After clicking the `Settings` link found in the dropdown, please locate the side navigation menu title ‘Personal settings’, it should be on the left hand side of the page. You’ll want to click on the SSH and GPG keys link. Here’s what I’m presented with after clicking.



On this page, you should find a green button that says ‘New SSH key’. Locate the button and give it a click.

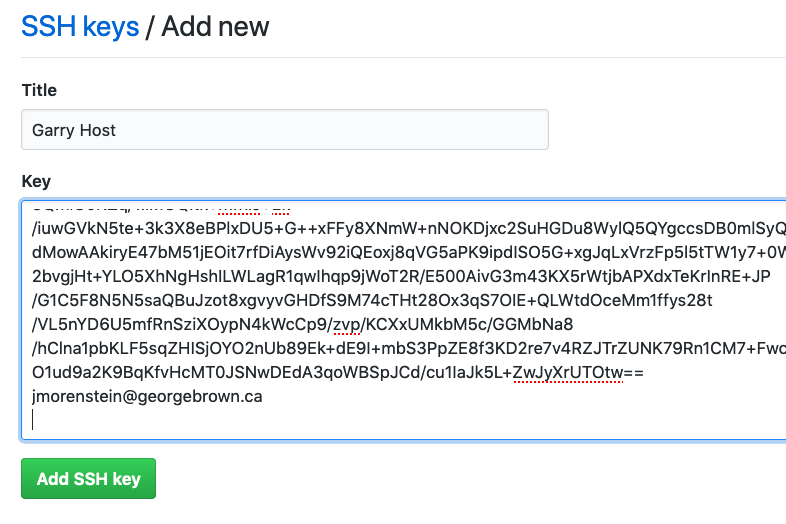


The button should bring you to a new page looking like this:

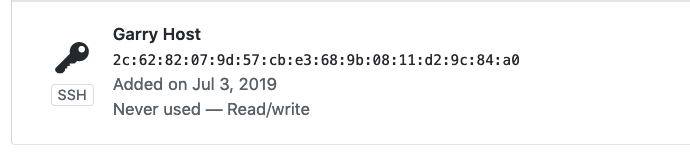


We now need to fill in some details. In the title section, please provide a title that accurately describes the location of your public key. In the key area, we’ll want to paste in our public key which was copied to the clipboard from the command line.

Please click the green ‘Add SSH key’ button once you’ve pasted in your public key. We should now be presented with the following page, if you run into any issues on this step, please see me if you aren’t able to resolve it. Don’t forget that you can also `cat` out the public key as I did in my terminal example or open the file with nano, vim or some GUI editor and copy from there.



After successfully Adding the key, you should see that your key has been added. It may look a bit different than mine, but that’s okay as you may have not used your key yet.



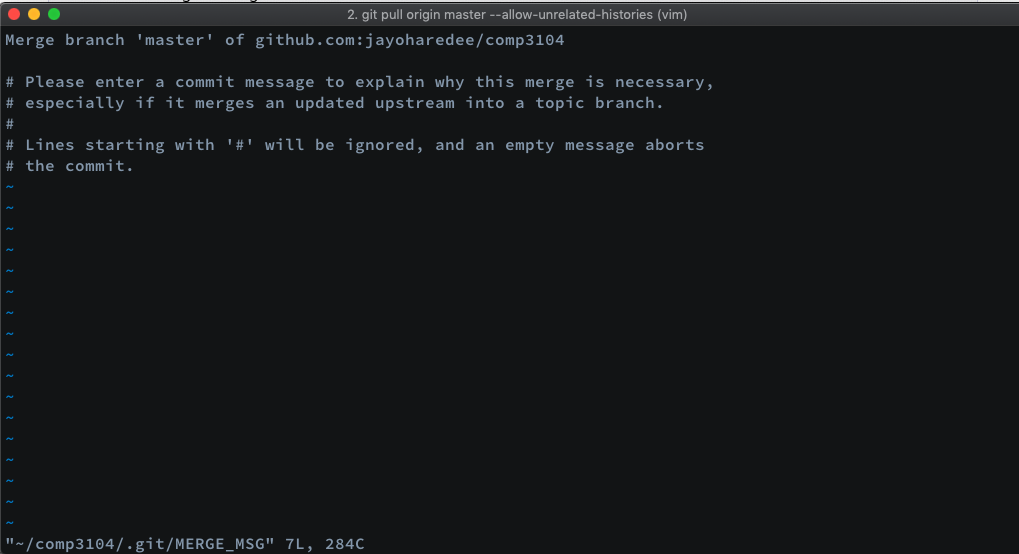
If everything has gone correctly, we should now be able to communicate with our remote repository for version control stuff. Let’s see if we can pull our license from our newly created Git repository. We’ll start by running the following command:

` git pull origin master --allow-unrelated-histories `

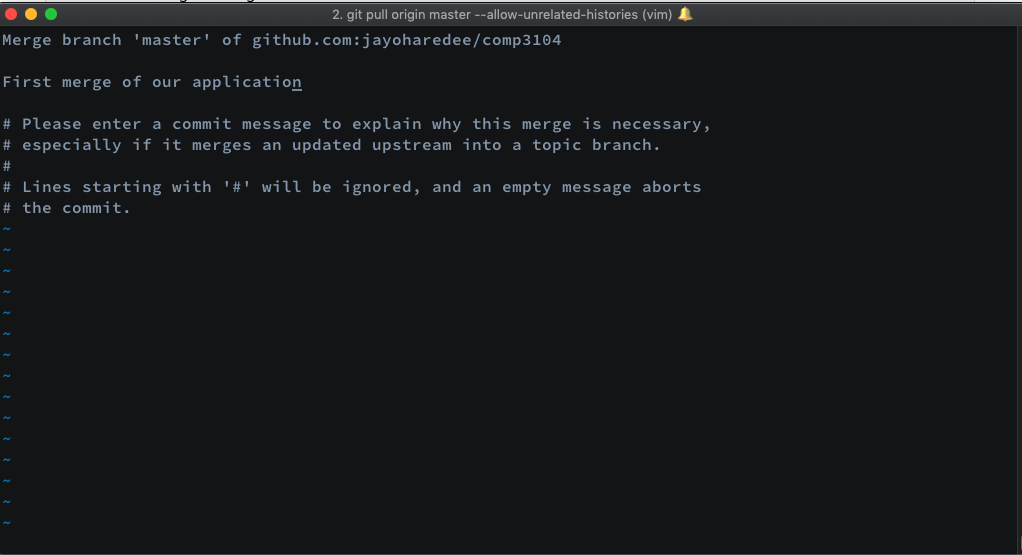
To unpack the above command a little bit, we are using git pull to pull from our origin source which we set with an earlier command. The branch we are currently working on is the master branch. The name kind of says it all, we’ll get into branching specifics a little later but just know for now, that the master branch typically refers to what should be deployed to the Production server. In this course, master will always reflect what’s in Production (the state of the application we want to serve our client traffic).

You’ll notice I’m using a switch, the switch has to be used in this instance because we populated our repo with a License when creating it, after we already had initialized a Git repo. This was done to illustrate that the Git history which we had on our local machine did not coincide with the Git history that was created with our new repository.

Now we have to perform our first merge. My default text editor is vi so I must first hit the ‘i’ key to put the editor into insert mode. Below you’ll see an image asking for us to provide a reason for the merge. I will show before and after photos of my merge.

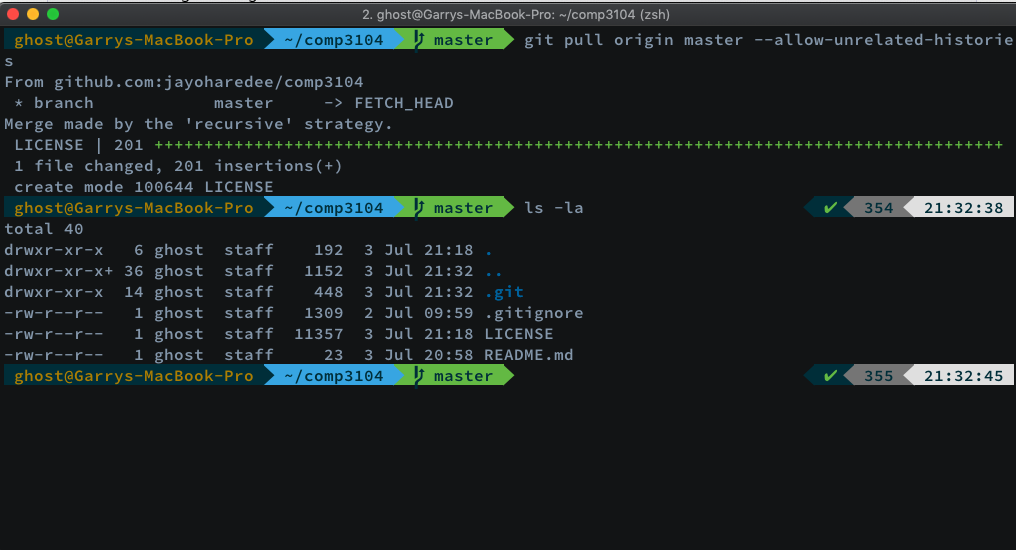


Here is my terminal after adding in a merge message:



Now that I want to exit, I’ll have to hit the `Esc` key followed by `:x` to save and quit. Your default editor may differ, if you have any questions that Google can’t answer please let me know.

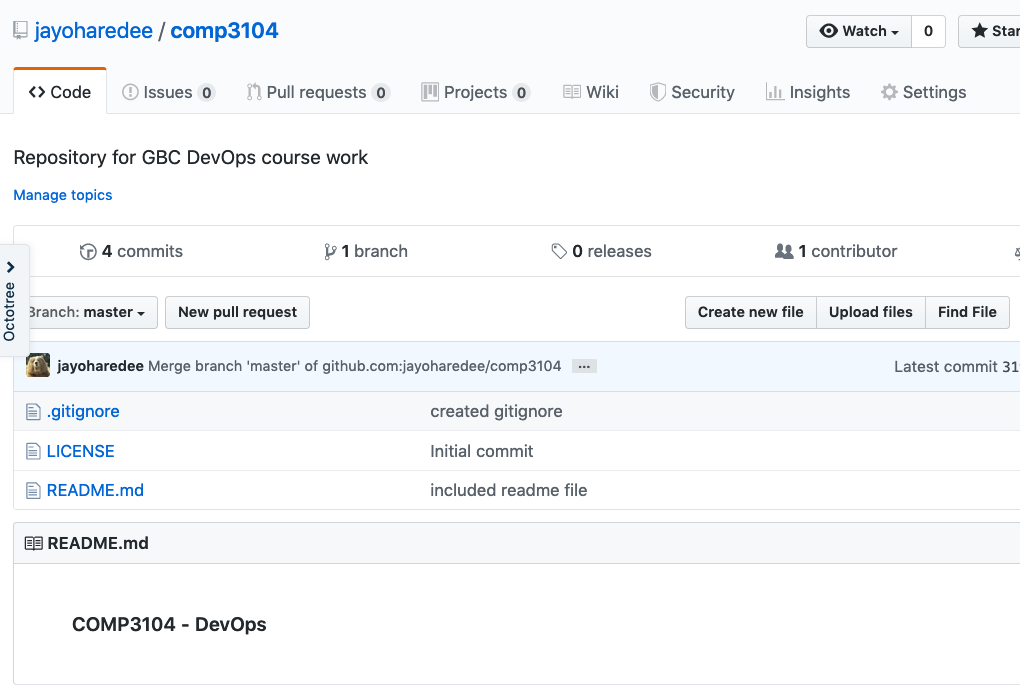
Below we can see that we now have our License file and that’s left to do is push our committed changes (the .gitignore and readme file) to GitHub.



And now let’s push our changes to the master branch:



If we go and check our GitHub repo for comp3104, we should see that our locally created files our now there.



You’ll see that we’ve successfully pushed our local changes and our course repo now reflects. You’ll also notice that the markdown file we created in last week's exercise is displayed at the bottom of the page.

Now that we have our local development environment successfully communicating with our public repo through SSH, we are all set up with basic Git configurations. Next up, we’ll be looking a look at the landscape to which a DevOps engineer commonly works in and some of the tools which will grant us the powers of a Continuous Integration / Continuous Deployment pipeline.