Lab 17: GitHub

1.1 Information

Topics: GitHub, Source Control

Turn in: You will submit your GitHub username via a lab "quiz" set up.

Contents

1.2   About Source Control, Git, and GitHub

What is Source Control?   Source Control (aka Version Control or Revision Control) is a type of software system that allows people to keep track of their changes over time, back up their work to a server, and make merging code between different people (or different machines) easier.

Storing your work on a server   By storing your work on a server on the internet, you can access your work from anywhere, simply by using a command like "clone" to pull down all the changes. No need to carry around a flash drive!

If you store your work on a source hosting service like BitBucket or GitHub, then you can also view your work through their web-based interface as-needed.

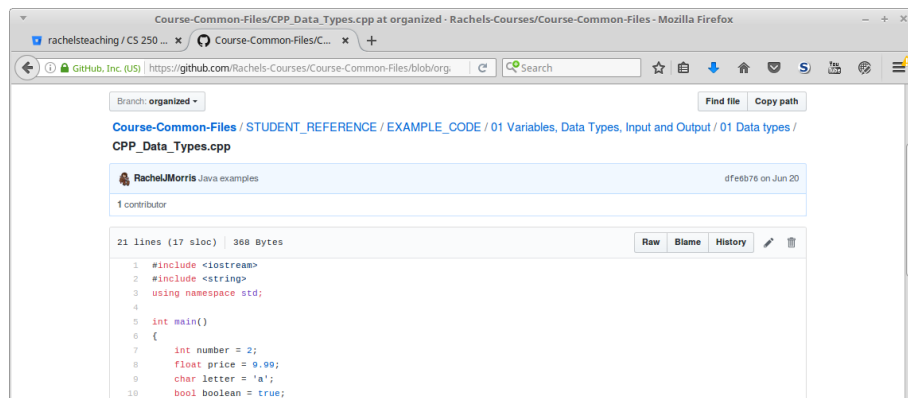Figure 1.1: Viewing a list of files on BitBucket

Figure 1.2: Viewing source code on GitHub

Keeping track of changes   Each source control solution also has features to let you review the changes to your code over time. By associating your changes with a "commit" and a "commit message", you can effectively make snapshots of your code over time.
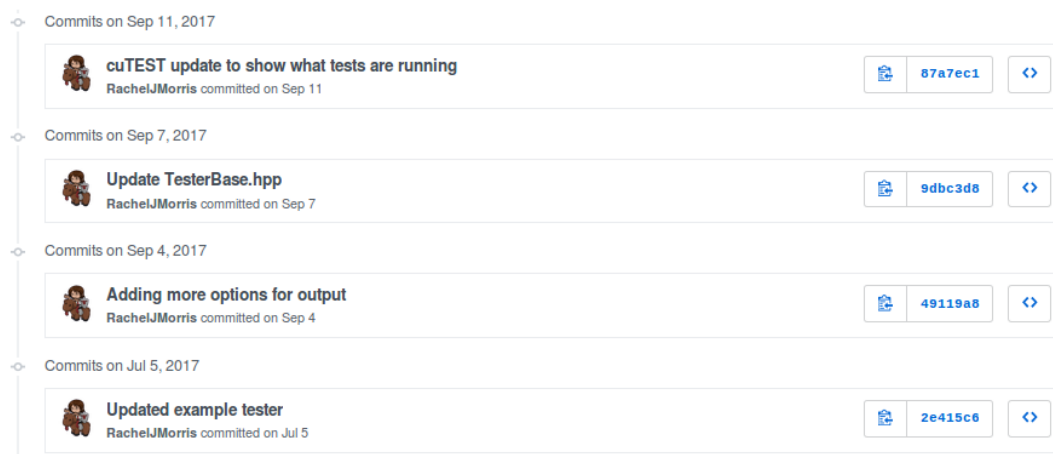

Figure 1.3: Viewing change history on GitHub

And, besides just viewing a list of changes, you can also do a "diff" to see exactly what changed between commits, line-by-line.

Merging   If you're working across multiple computers, or working with other people, you may have different versions of the code on different machines. With a source control program, it will take care of merging in different changes to the same file - and if it can't figure out how to automatically merge files, it will give you some interface to do it manually in an easier manner.
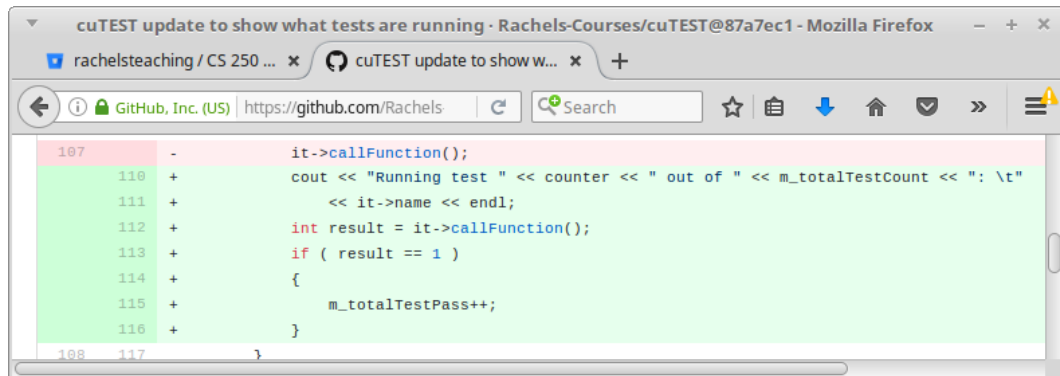
Figure 1.4: Viewing line changes in GitHub - Red for lines removed, green for lines added

In the case above, two people edited the same code around the same time. Once Person A pulled the latest code from the server, the knock knock joke was already committed and pushed up. Git pulled the latest code and tried to merge, but couldn't figure out which was the correct version for this function, so instead it added markers around the part it didn't know how to merge on its own. That way, the programer can manually decide what to save and what to remove.

What are some version control systems?   Some common version control systems are:

Git

Git is a popular source control system for open source projects and work that doesn't use Microsoft products (usually, Microsoft shops use TFS). Git was created by the maker of Linux, Linus Torvalds, and it is open source.

https://git-scm.com/

```
1    #include <iostream>
2    using namespace std;
3
4    void DisplayMessage()
5    {
6    <<<<<<< HEAD                                                    Changes A
7        cout << "Hello! How are you?" << endl;
8        cout << "I'm fine, how are you?" << endl;
9        cout << "I'm also good, thanks!" << endl;
10   =======
11       // Person B changes                                          Changes B
12       cout << "Knock knock" << endl;
13       cout << "Who's there?" << endl;
14       cout << "Orange" << endl;
15       cout << "Orange who?" << endl;
16       cout << "Orange you going to let me in?" << endl;
17   >>>>>>> baf54d4efd84f54039780c2bc897760a1afbf148
18   }
19
20   int main()
21   {
22       DisplayMessage();
23
24       return 0;
25   }
26
```

Figure 1.5: Merge conflict flags added to the code by Git

Mercurial

Mercurial is another popular source control system, built by Atlassian (who also make Jira and BitBucket). Mercurial is available as Free software under GNU GPL v2.

https://www.mercurial-scm.org/

TFS

Team Foundation Server (TFS) is Microsoft's source control solution, which is commonly used at companies that use Microsoft tech, such as C#, ASP.NET, and others.

https://www.visualstudio.com/tfs/

Subversion (SVN)

Subversion is an older source control solution, and (in my opinion) it is less robust than the above three. However, it is still used in various places, so you might run into it sooner or later.

https://subversion.apache.org/

What are some source hosting sites? You can always host your own source control server on your local machine or on a server you run, but you can also host your code on a hosting service.

Many hosting services allow free hosting for public repositories, so they are open source-friendly. Otherwise, they may charge for private repositories, such as for hosting a business' product repositories. It depends on the service!

GitHub GitHub allows you to host Git projects. It is free if your repository is public, but costs if you have private repositories.

Bitbucket Bitbucket can host Mercurial and Git projects. It is free for public and private repositories. Private is free for small teams of up to 5 users. Bitbucket is ran by the creators of Mercurial, Atlassian.

(Usually I host my open source stuff on GitHub and my startup's private repos here.)

CodePlex CodePlex is Microsoft's hosting solution for open source projects. It supports Mercurial, TFS, SVN, and Git.

Sourceforge Sourceforge is a popular hosting site for open source projects. It supports SVN, Git, Mercurial, and some other source control systems.

What are we doing? For this lab, you will be creating a GitHub account and using Git to upload your programs from this semester to a CS 250 repository. This way, it will be available for you to reference in the future as needed, but also will be visible to potential employers. It is good to keep a portfolio of your work on GitHub so that potential employers can see what you've worked on. Your code doesn't have to be perfect or beautiful, but it is good to see what you've accomplished.

1.3   Getting started

1.3.1   Creating a GitHub account

First, create a GitHub account at https://github.com/ . You will need to specify an email address, username, and password. You can use your school email or your person email - whichever you're less likely to forget. :)

When it asks what plan you'd like, select the free once.

Once you're finished, you will have a profile at github.com/YOUR-NAME You will be able to create repositories from this directory.

1.3.2 Creating a CS 250 repository

From the web interface, you will create a repository for the CS 250 class.

What is a repository?   You will generally create one repository per project that you're working on. For example, id software has the following repositories...

- Quake III Arena
  https://github.com/id-Software/Quake-III-Arena

- DOOM 3
  https://github.com/id-Software/DOOM-3

- Wolfenstein 3D
  https://github.com/id-Software/wolf3d

So essentially, each game has its own repository.

In this lab's case, you will throw all your CS 250 projects into one repository, but for future projects, you could put each project in its own repository.

Creating a repository   From your GitHub profile page, github.com/YOUR-NAME, click on the **Repositories** tab, then click the **New** button.
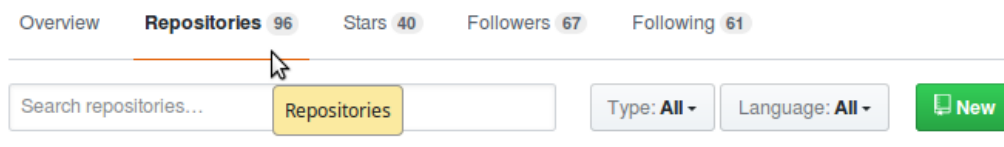
When creating the repository, make sure to set the following...

- Repository name: CS250

- Description: Data structures, Fall 2017

- Access: Public

- Initialize this repository with a README (check!)

- Add .gitignore: C++

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

    RachelJMorris ▾   /   CS250   ✓

Great repository names are short and memorable. Need inspiration? How about **musical-robot**.

Description (optional)

My Data Structures projects - Fall 2017

Public
Anyone can see this repository. You choose who can commit.

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

☒ **Initialize this repository with a README**
    This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **C++** ▾ | Add a license: **None** ▾   ⓘ

Create repository

Once you've set it up, click **Create repository**. It will then take you to the repository page. Leave this open, we will return to it later.

1.3.3   Installing Git

Next, you will need to set up Git on your machine. The offical Git website is at: https://git-scm.com/

Lab computer: If you're working on a lab computer, they should already have Git installed.

Windows: Go to the Downloads page and select the Windows download. Download either the 32-bit or 64-bit depending on your computer.

Linux: If you're using a Debian-based Linux distro, you can install it via the package manager or command line.

Just search for "git" in the package manager...
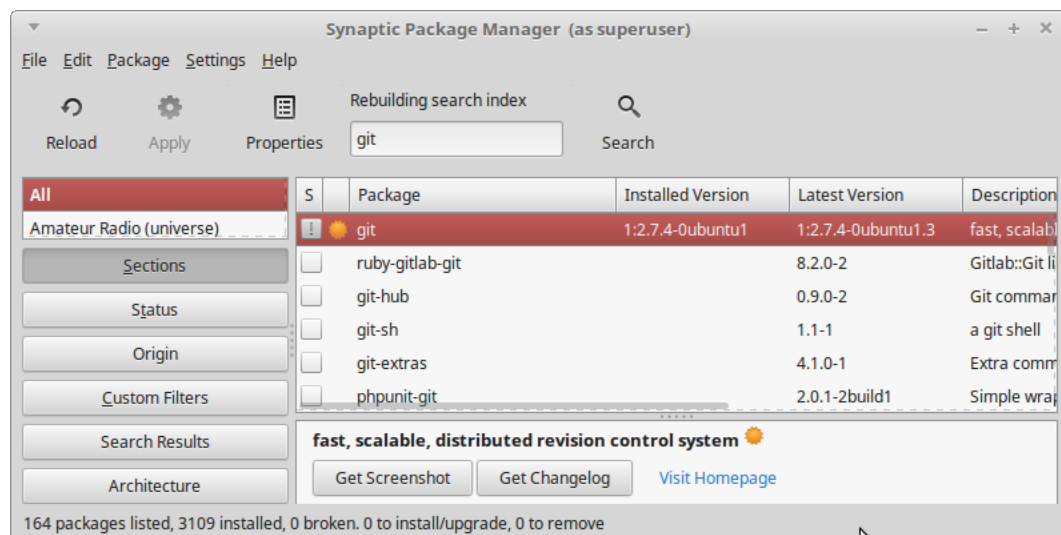
Or open the terminal and use...
```
sudo apt-get install git
```


Mac: I don't have a Mac computer I can test this out on. If there's a package manager for Mac, try to install it from there first. Otherwise, download the Mac package off the Git website.

1.3.4   Cloning the repository

Next, locate a place on your computer where you will want your CS 250 repository folder to be located. Right-click here to open up **Git bash** (Windows/Mac?) or the terminal (Linux). We will be using Git from the command line, but there are also GUI interfaces for it.
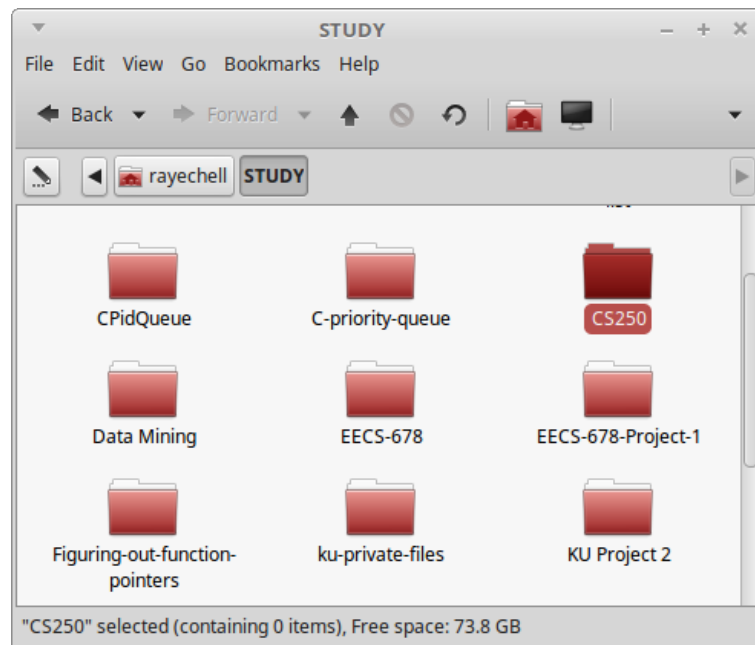
Back on your profile page on the GitHub website, click on **Clone or download**. Make sure HTTPS is selected (you can set up SSH later if you want), and copy the URL given.
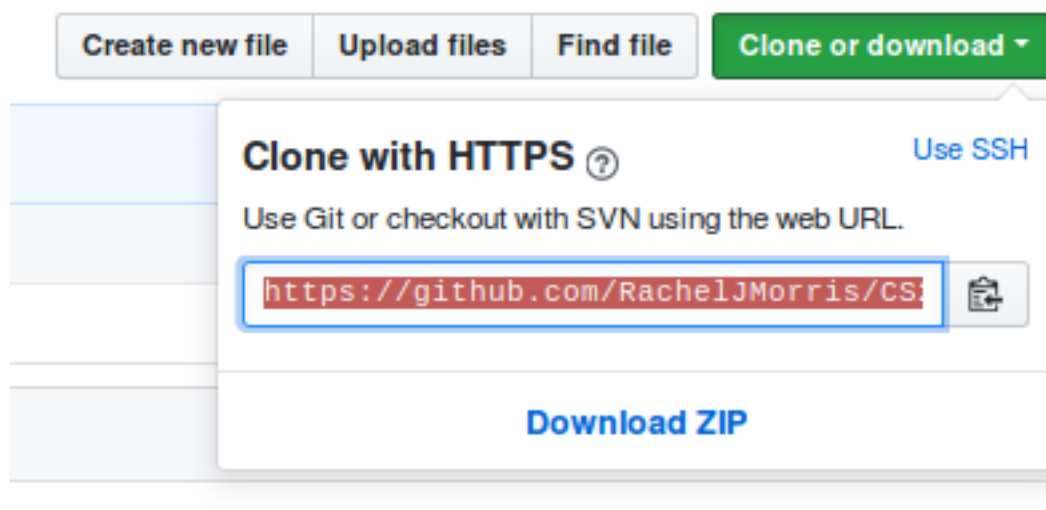
Back in the terminal, type "git clone" and then paste in the URL.

This will create a folder in the directory you chose.

Now you can do all your work in this folder, and any changes that you commit can be pushed to the server.

1.4   Moving your files over

Now you will move over your project files from class this semester into this repository folder. This should include your lab and project files. For example:

```
CS 250/
 └──Labs/
      ├──Lab 01 - STL/
      ├──Lab 02 - Exception Handling/
      ├──Lab 03 - Static Array Wrapper/
      ├──Lab 04 - Testing/
      ├──Lab 05 - Dynamic Array Wrapper/
      ├──Lab 08 - Templates/
      ├──Lab 09 - Polymorphism/
      ├──Lab 11 - Priority Queue/
      ├──Lab 12 - Dictionary/
      └──Lab 14 - Recursion/
 └──Projects/
      ├──Project 1 - Linked List/
      ├──Project 2 - Stacks and Queues/
      └──Project 3 - Binary Search Trees/
```

1.5   Add, Commit, Pull, Push

1.5.1   Add files

Now, navigate to the base folder of the repository (The CS250 folder) and open Git bash or the terminal.

There are several ways you can add files...

```
rayechell@rayechell-GP60-2PE ~/STUDY/CS250 $ git add *.cpp *.hpp *.txt *.csv
```

- `git add FILENAME` - Add a single file

- `git add *.cpp` - Add everything that is a ".cpp" file in all subdirectories.

- `git Labs/*` - Add ALL changes in the Labs folder and subdirectories in Labs/

- `git add .` - Add ALL changes in all subdirectories.

Generally, you shouldn't add any .exe files, or other files that are generated each time the project is built. There's no reason to keep these in source control because they're auto-generated.

You can store your project files if you'd like, though Visual Studio project files tend to be big.

If you want to stick to the basics, just commit any .cpp, .hpp, .txt, and .csv files with the command I used in the screenshot:
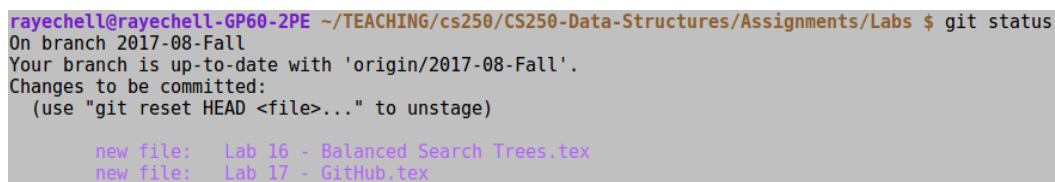
<div align="center">

`git add *.cpp *.hpp *.txt *.csv`

</div>

The * is a wild-card, meaning "anything can go here". Since each item is *.EXTENSION, it will be any file as long as it matches the ending extension.

If you want to double-check what you're adding, type:

<div align="center">

`git status`

</div>

and it will give you a list of files waiting to be committed.

Figure 1.6: The CS 250 lab specs waiting to be committed. So meta.

1.5.2   Commit files

Next, you will need to commit the changes. Each time you update files, you will need to `add` them, and then make a new `commit`. A commit can have multiple adds.

To commit your changes with a message, use:

<div align="center">

`git commit -m "Adding my CS 250 files"`

</div>

It will display a list of the items being added. Since you're adding everything at once this time, it might be a huge list!

1.5.3　Pull and Push files

Making commits only stores the changes locally on your machine. When you're ready to push your changes up to the server, you will use the `push` command.

However, if you're working across multiple computers, or with other people, it would be good to practice pulling prior to every push. `pull` pulls all the files from the server and performs a merge on anything that's changed.
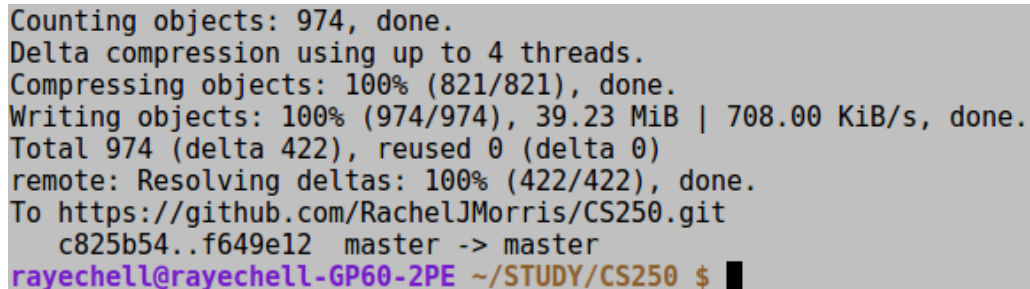
<div align="center">

`git pull`

</div>

Right now, pull will just say that everything is up-to-date because nobody else is working in your repository. After the pull, make sure to push your changes.

<div align="center">

`git push`

</div>

It will have some messages about setting up defaults. There are a lot of extra features of Git that you can study up on later, but these are just the basics!

It will push all updates to the server and display the status:

```
Counting objects: 974, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (821/821), done.
Writing objects: 100% (974/974), 39.23 MiB | 708.00 KiB/s, done.
Total 974 (delta 422), reused 0 (delta 0)
remote: Resolving deltas: 100% (422/422), done.
To https://github.com/RachelJMorris/CS250.git
   c825b54..f649e12  master -> master
rayechell@rayechell-GP60-2PE ~/STUDY/CS250 $ ▮
```

Figure 1.7: A successful push

1.6 Viewing on the web interface

Once your changes have been pushed, you can now view them on the web interface:

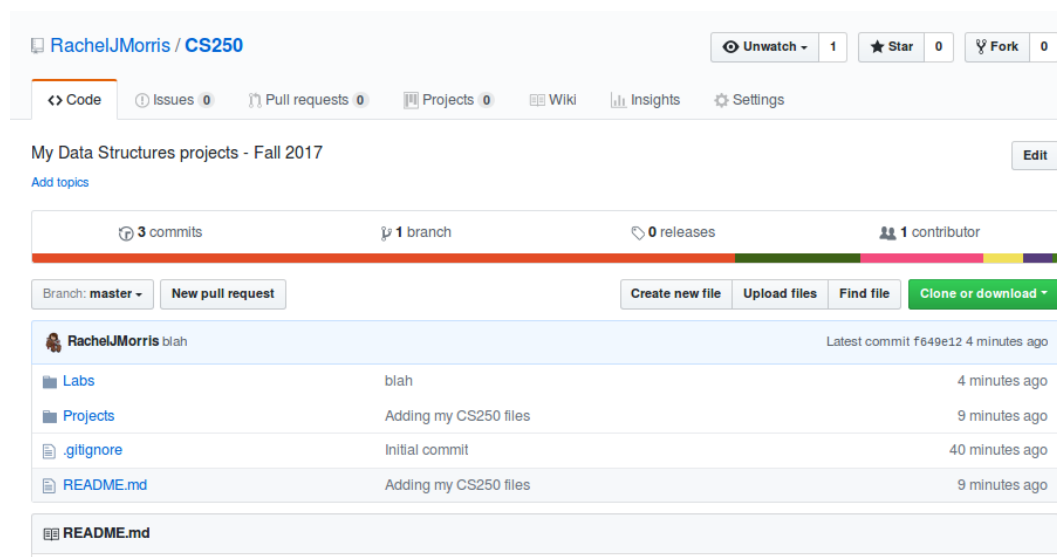$$\texttt{https://github.com/YOUR-NAME/REPOSITORY-NAME}$$

Figure 1.8: Repository on the web

1.6.1 View history

Click on the **commits** link from the web interface.

Figure 1.9: Repository on the web

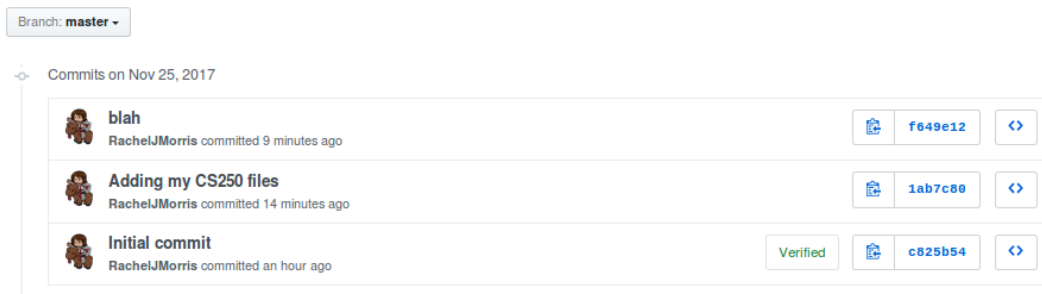It will bring up a list of all the commits done in this repository.

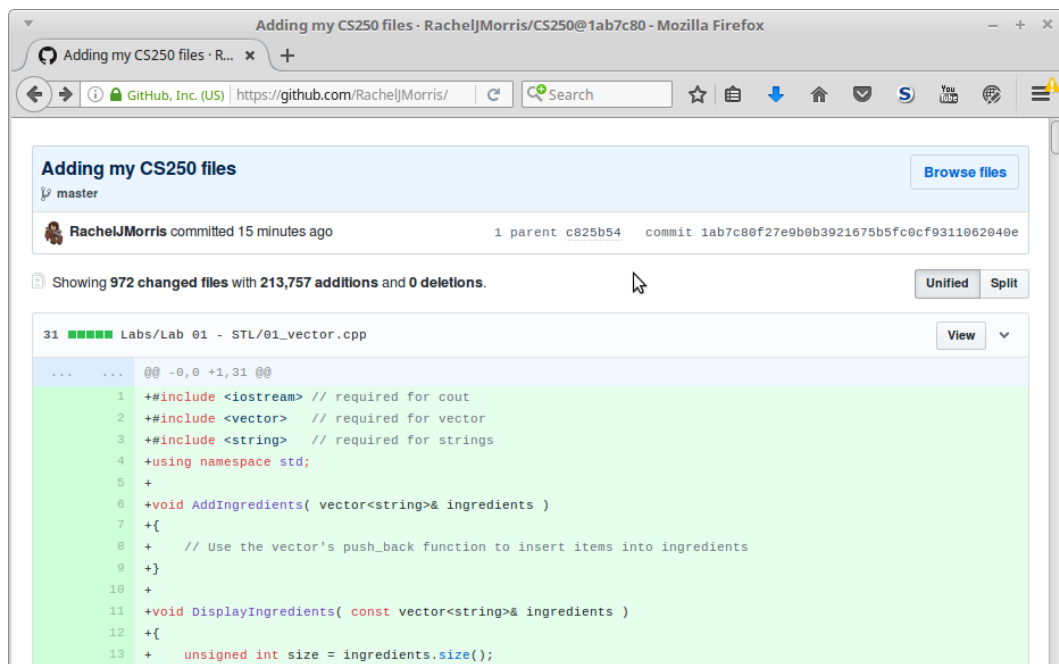Figure 1.10: Viewing all commits

You can click on a commit message to view all the changes for all the files.

Figure 1.11: Viewing one commit