Git Basics
with GitHub

Creating a Repository

From the GitHub web interface (https://github.com/YOUR_USERNAME), click on the **Repositories** tab. There will be a green button labeled **New**.

Repository name:

Your repository's name; this will be part of the URL for the web interface, and the name of the folder when you clone it to your hard-drive.

Description:

Extra data for people viewing it on the web.

Public/Private:

It is free to make a public repository, but everybody can see the code. (It is good for creating a portfolio of your work, though!)

Initialize this repository with a README:

Usually you will want to check this. A README file will be displayed on the web interface automatically when your repository is viewed.

Add .gitignore:

Set this to whatever language you will be using in the repository. It will create a file that tells Git to ignore certain filetypes – such as the compiled binaries, and temp files.

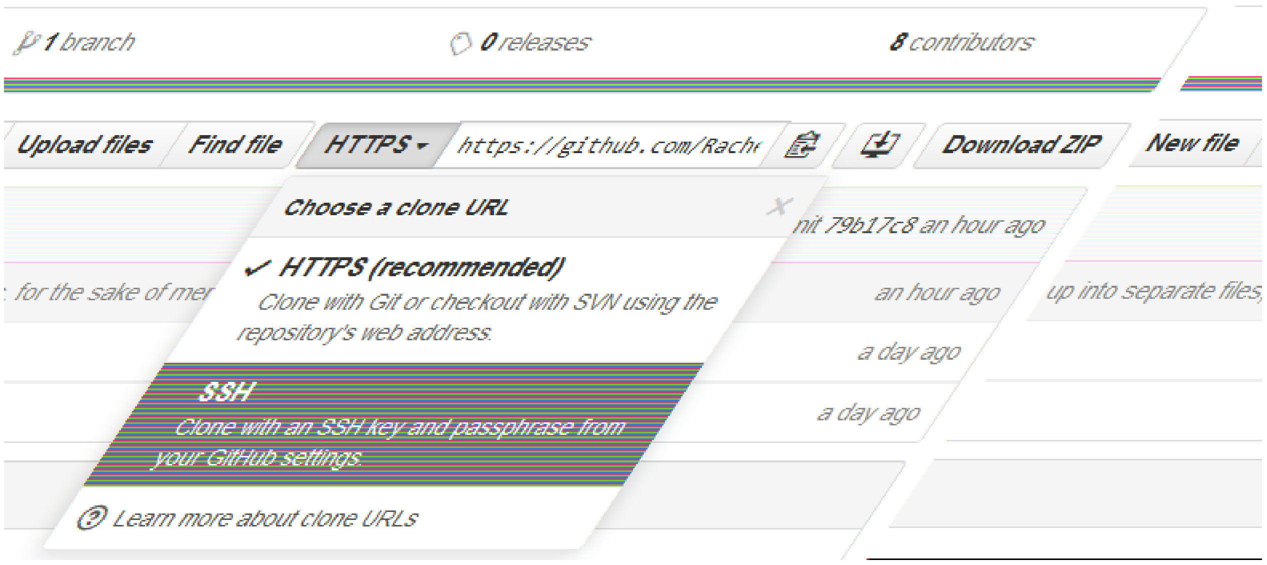
Add a license:

Optional, you may want to review license terms for common software products.

Clone the Repository

After you've created a repository on GitHub, you will want to clone it to your hard-drive so you can work on your machine.

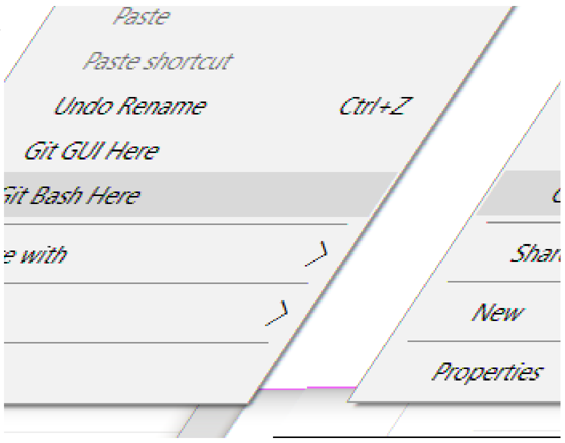
At the web interface of your repository, there will be a dropdown menu with two options: HTTPS and SSH.



If you have GitHub Desktop installed on your machine and set up, you can use SSH. Otherwise, the SSH method takes some extra set up (GitHub has a tutorial). If you want to keep it simple, just use HTTPS.

There is a URL next to this dropdown box – highlight it and copy this URL.

On your hard drive, navigate to a place where you want to store your repository; for example: C:\Projects . To open the GitHub command line, right-click in the empty space and select Git Bash.

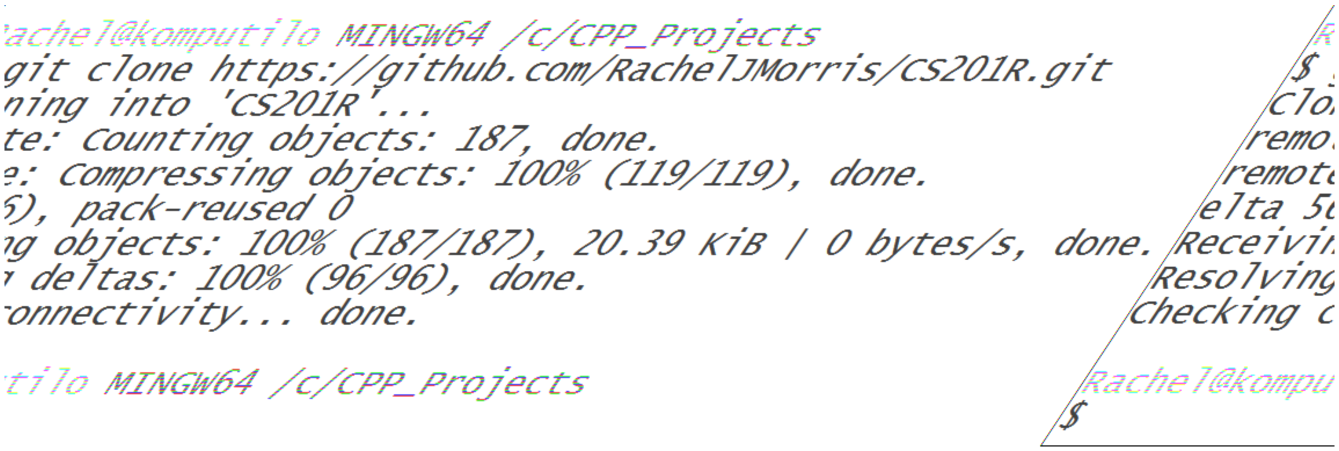


It will open up the command line.

Type in:

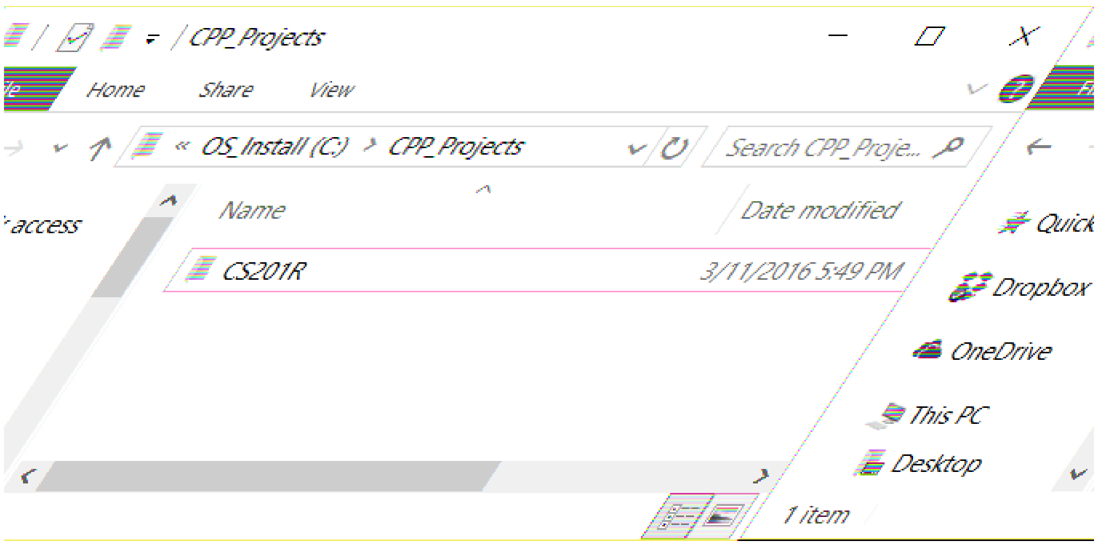
`git clone`

Then, paste the URL from GitHub. You can paste in this window by right-clicking and selecting **paste**. Hit enter and your repository will be brought down and put in its own folder.



```
Rachel@komputilo MINGW64 /c/CPP_Projects
$ git clone https://github.com/RachelJMorris/CS201R.git
Cloning into 'CS201R'...
remote: Counting objects: 187, done.
remote: Compressing objects: 100% (119/119), done.
elta 5 5), pack-reused 0
Receiving objects: 100% (187/187), 20.39 KiB | 0 bytes/s, done.
Resolving deltas: 100% (96/96), done.
Checking connectivity... done.

Rachel@komputilo MINGW64 /c/CPP_Projects
$
```



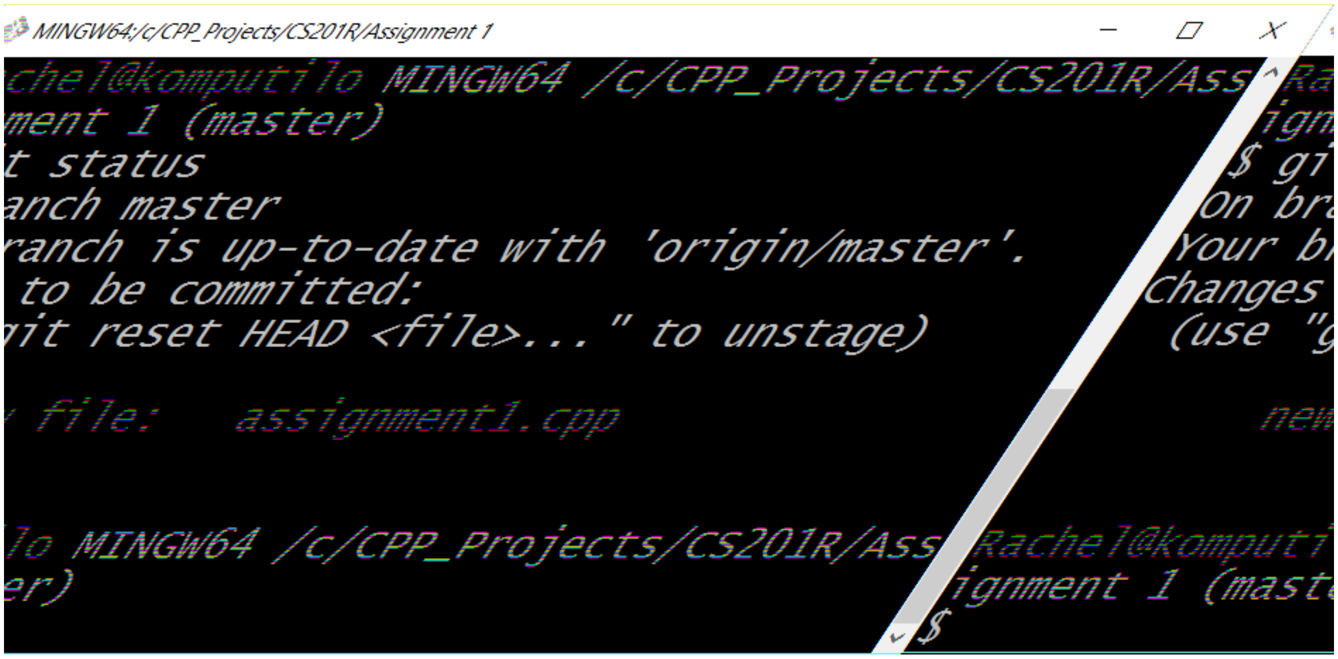
Now you can create a project or add files within this folder.

Committing Files

Within your project folder, add a new folder such as “Assignment 1”. Within the folder, move your source files (.cpp and .hpp) from your first assignment into it.

You can bring up Git Bash within any folder of your repository to set commands. Open up Git Bash (make sure it is within your repository directory) and type:

```
git add .
git status
```



```
Rachel@komputilo MINGW64 /c/CPP_Projects/CS201R/Ass
ignment 1 (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   assignment1.cpp


Rachel@komputilo MINGW64 /c/CPP_Projects/CS201R/Ass
ignment 1 (master)
$
```

The “git add .” command adds any files within the current directory to a list of files that have changed, that we will commit to the repository later. You can also specify specific filenames like:

```
git add assignment1.cpp assignment1.hpp
```

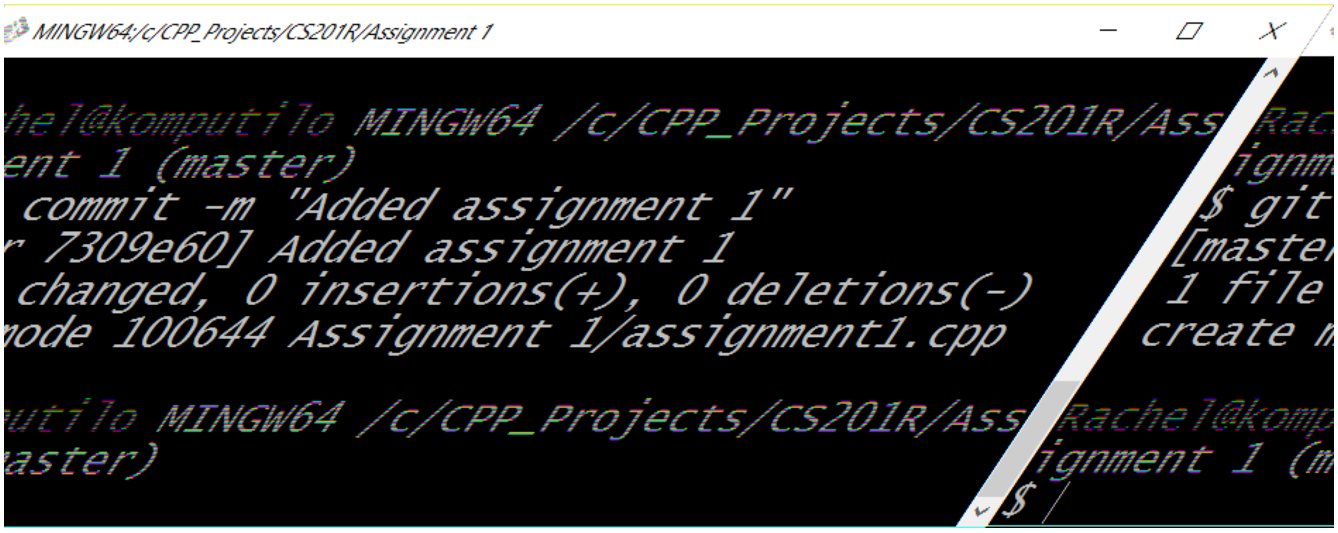
The “git status” command will show you a list of files that have changed and are waiting to be committed.

With Git, we can make a snapshot of changes to our code at any point in time. To do so, we need to create a **commit**. Every commit will save the files that you've added at that point in time. Later on, you can go back and view all the changes along the way.

To make a commit, type in:

```
git commit -m "Added assignment 1"
```

Where after the -m is your **commit message**, which will show up in the Git log and help you locate specific changes.



```
hel@komputilo MINGW64 /c/CPP_Projects/CS201R/Ass  Rac
ent 1 (master)                                    ignm
 commit -m "Added assignment 1"                   $ git
r 7309e60] Added assignment 1                     [maste
 changed, 0 insertions(+), 0 deletions(-)         1 file
node 100644 Assignment 1/assignment1.cpp          create m

utilo MINGW64 /c/CPP_Projects/CS201R/Ass  Rachel@komp
aster)                                    ignment 1 (m
                                          $
```

A commit will make a snapshot on your local machine, but won't automatically be added to the server. To push your changes to the server, use:

```
git push -u origin master
```

Later on, you can create different code **branches**, which could be useful when working with other people on one project, so that everybody is working on a different branch for their feature. For now, we are just using the **master** branch.

Once your code has been pushed, all the changes should appear on the GitHub website. Note that you can commit multiple times before pushing.

Pulling and Merging

If you are working with other people, or just working on different machines, you will eventually need to pull the latest changes from the GitHub server to your local machine. Sometimes this can cause merge conflicts.

1. Commit your changes

Before you get the latest set of code from the server, you need to commit any changes to files that have been modified.

```
git commit -m "Changes"
```

2. Get latest code from the server

Secondly, you will need to get any changes on the server. You can do this with:

```
git pull
```

Git will automatically attempt to merge the files together. Usually it is pretty good at merging files, so long as you and another person weren't working on the same code area at the same time. After Git merges, it might ask you to create a commit message for the merge. You can push after this.

3. Handling merge conflicts

If Git gives you a message about there being merge conflicts, it will tell you which files had conflicts. You will want to open up these files in your editor so you can manually fix them.

Git will add flags within your code file that look like this:

```
<<<<<<< HEAD
#include <vector>
#include <string>
=======
#include <locale>
#include <fstream>
>>>>>>> 7a270323f9ee83bf830bc5130681e46643ec5c38
```

In this case, it will display two versions of the code, split by the ======= line. It is up to you to decide whether you want one, the other, or both. Manually fix the code, then remove the flags that Git added to your code. Afterwards, do a new commit.

4. Pushing your changes

Finally, after you have merged and committed, you can push your changes to the server. Use:

```
git push -u origin master
```

and the changes should go up to the server.

Your changes should now be visible on the web interface of GitHub, and any changes you pushed will now be pulled down for anyone who runs the "git pull" command.