# Github & Git Prep for Wednesday

These next couple of steps are a mishmash of steps that need to be completed individually to get us ready for the lab on Wednesday. Most of you have done some of these things already, so if you have, just make sure to read through the step to ensure you did all the steps that are here.

## Step-1

Signup for a github account.

Easiest way to do this is by going here:

**http://education.github.com/pack**

If you already have a github account, you can still get the pack (and you should!), but you can also just go to the next step. Some of these are just awesome technologies that might be of use later when you are writing programs to take over the world. One thing you’ll need to do for certain is to verify your email address. You can check to see if your email is verified by going to the github settings clicking on the icon on the upper right, going to *Settings*, and then clicking on *Emails* in the Personal Settings list on the left. The other thing you should do is give youself an icon, it can be the same one as from basecamp if you’d like. You can also change this in settings.

## Step-2

Now we are going to make sure that you have a version of git on your computer

Let’s get git. This time around we are not going to use eclipse to help us manage things, so what I’ll be supporting for you this time around is the command line. For now, give it a try. If you are still unsatisfied, then you can try to move to a GUI version of your choice, but do this command line one for now.

Go to <https://git-scm.com/downloads> (or search “download git” via google) Pick your operating system of choice. If you are on a lab computer, you may need to select the portable version once you get to the windows page.

## 

## Step-3

Get git installed.

After downloading the installer open it up and install it on your computer. All of the default options should be fine in this situation, though you can change some options if you feel necessary. Make sure though that the **option to commit in Unix Line Endings is enabled**. Keep that preference. That will help sure that your files will play nice with everyone else. Once you have it installed, go ahead and open it, and let’s change some user preferences in git.

## Step-4

To streamline your git activity, let’s add some of your credentials to git preferences. I took most of this from the git-scm book, but you’ll want to **set the git preferences on all your computers.**

With the git command line open, type the following commands, using your own credentials

git config --global user.name "John Snow"

git config --global user.email winteriscoming@example.com

Vim is the default text editor that is used in the command line. If you aren’t comfortable or have never used vim, you can setup other unix text editors instead. So to use emacs you’d write:

git config --global core.editor emacs

If you’re like me, you’ll want to use something like notepad++ or possibly some other text editor that you love. In those cases you’d have to provide the entire path.

git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -nosession"

If you want to check your preferences, you can just ask

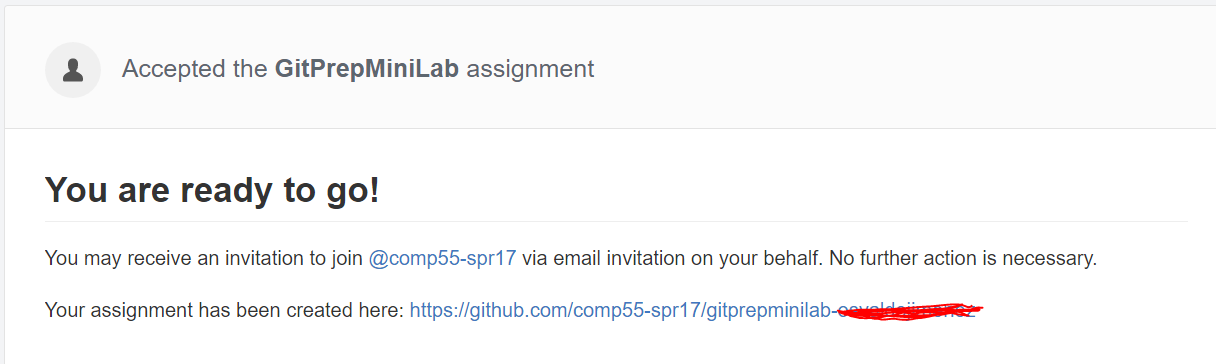
git config --list

## Step-5

Now with your git setup and your github account ready visit this URL:

**http://j.mp/159gitprelab**

Make sure to accept any of the authorizations that are present and to Accept the assignment. This is going to have you enter the comp159 organization (not the 55 org listed in the picture, but similar). If you have already signed up for an account it may ask you to re-enter your credentials. Once you have clicked on all the admin processes and you’ve reached the place where you get the message below (with your github username down below), you can move on to the next step. Leave this window open as we’ll need to access it later.



## Step-6

Use the git command line tool to reach the appropriate directory. For this portion, you’ll want to go to a particular location using the git command line to go to a directory where you would like the folder containing the unity project to be placed in. If you’re not sure how to use the command line, feel free to ask, but easiest commands to learn are:

cd (changes the directory to the one specified afterwards, to go up use cd..)

ls (list the files in a directory, if you want to see all files, say ls –a)

mkdir (makes a directory with the name you specify afterwards)

And press the tab key to autocomplete any folders and folder paths as you type so that you don’t misspell things as you go along. Another other option you have is, if you are on your own computer, use windows explorer to get to the directory where you want to download the project, and then right click and select *Git Bash Here*

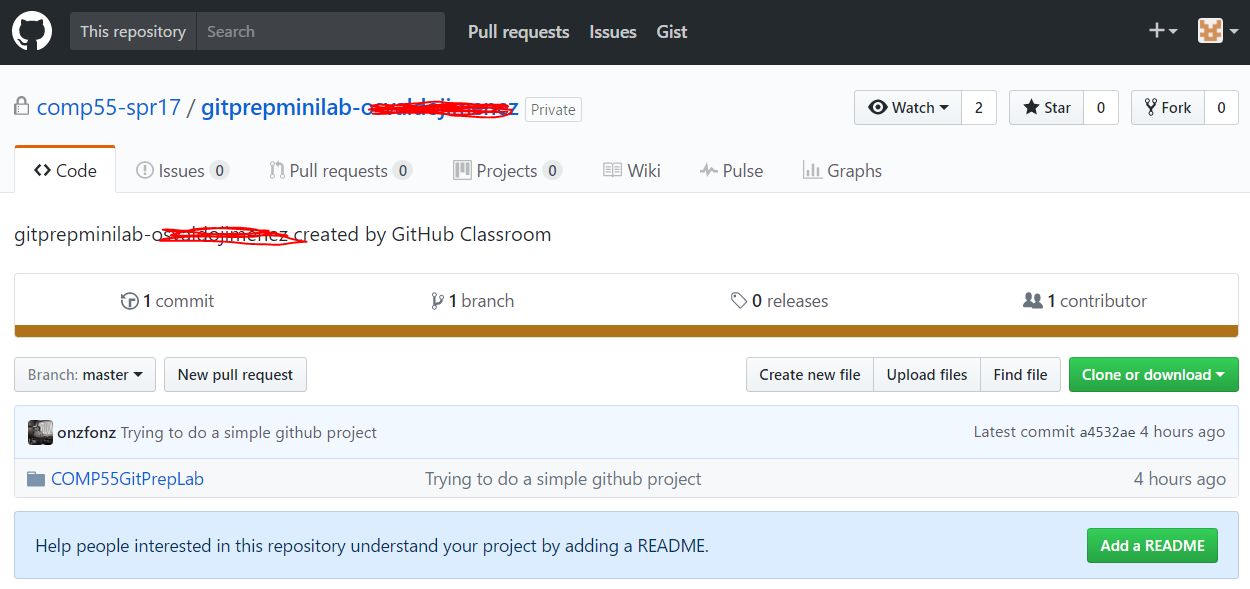
## Step -7

Clone the repository from github to your machine

**git clone THE\_URL\_YOU\_COPIED\_HERE**

How do you get the correct URL? It’s listed on the repository’s github page.

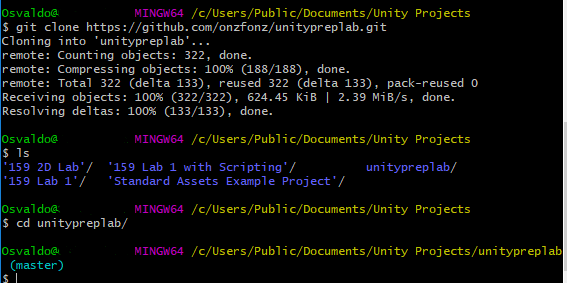
If you’re not sure where that is, just head back to github, click on the repository that you just made, and then click on the green download link. If you needed to log back into github and can’t find the repository they will be listed on the right once you login to github. Once you click on the correct repository you’ll eventually see a page like this:



The URL will be listed and you can either type it or copy it, which you can do using the clipboard . Once you do this you can go back to the command line go to the directory where you want the git folder to be in and type the command above, right clicking to paste the URL in. Once it finishes, move into the unitypreplab directory that was just created.

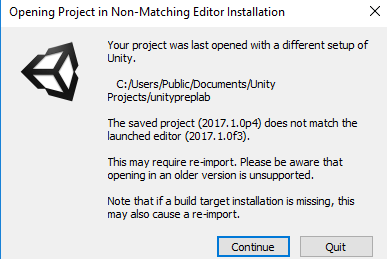
## Step-7

Do a sanity check. You should see that the command line now shows a highlight showing you that you are on the master branch, which would be in a different syntax color: notice how the end of the prompt ends with master, that lets you know that you are on the master branch of a git project. You can take a look at the files either in the command line or via your explorer of choice.



## Step-8 – Running the downloaded project

Now with all of the files downloaded, you can open up unity. Select the open project option after opening up unity, and go to the directory where you downloaded your file and open it up. You may get a warning about it being created in a newer version, just say continue to re-import the project.



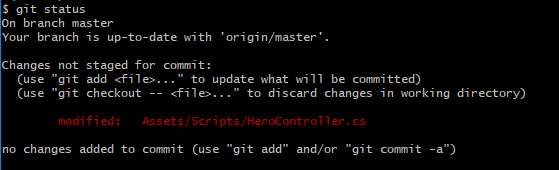
Once the project is opened, open up the Main.unity scene via *File->Open Scene* and select the **Main.unity** file to open. You can then run the project, which doesn’t do anything really.

## Step-9 Making changes and sending them upwards

For this mini-lab, we’re only interested in getting you used to the mechanics of making a simple change and uploading that change to github. So here’s what I’m going to ask you to do.

Change line 33 in HeroController.cs so that the comment says “FixedUpdate used when making physics calls”, and change the magic number that is on line 38 from 6 to 5.

Once you do this, save the files, save the scene & save the project. Working with git and unity is a little bit more cumbersome, unity’s had a history of not always playing nice with files or knowing how to reload when files themselves change, so it would be safer for you to when changing files and committing that you **close up unity before issuing git commands**. Once unity is closed head back to the command line and type **git status** to see what’s changed.



Notice the red text and the message that git gives us. It tells us that HeroController.cs has changed. However, it also says that it is **not staged for commit**. Unstaged changes are changes that git has yet to put as part of the next commit. When they are staged, git knows that on the next commit, it will record those changes as being different. While I typically stage most of my changes, one reason that I’ve heard for not staging changes is in order to make multiple commits so that you split apart multiple changes that you’ve made. Leaving files unstaged won’t make them a part of the commit, so you can commit multiple times and add the appropriate files to address each bug or feature for example. However, we will for the most part tell git to commit all changes. When files have changed in git, git wants to give them a new version number. To give or to save these files so that git can reference them in the future, you’ll want to do what’s called **“commit”**. To commit the files, type:

git commit –am “your message explaining the changes you made here”

Committing will save the changes to the local repository. The –a switch will commit all modified files, both staged and unstaged, while the –m allows you to type your message as part of the commit, instead of opening up that text editor option that we had before. If you have new files that git has yet to keep track of, then you must say:

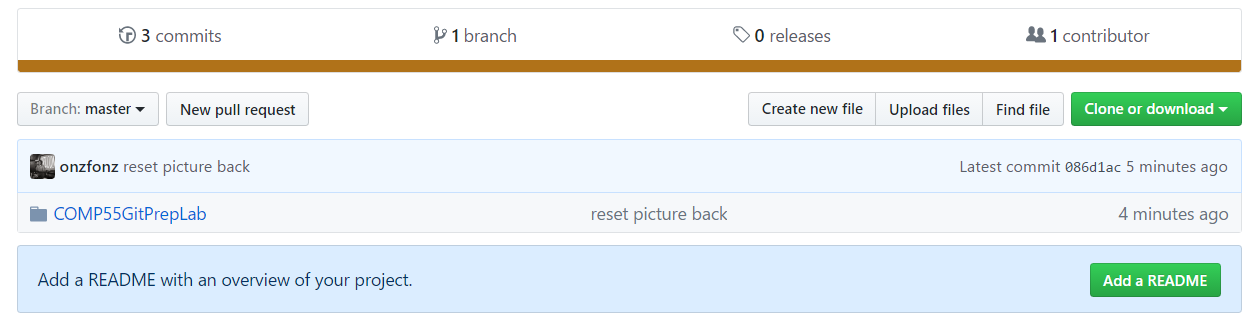
git add NAME\_OF\_FILE\_OR\_DIR

## Step-10

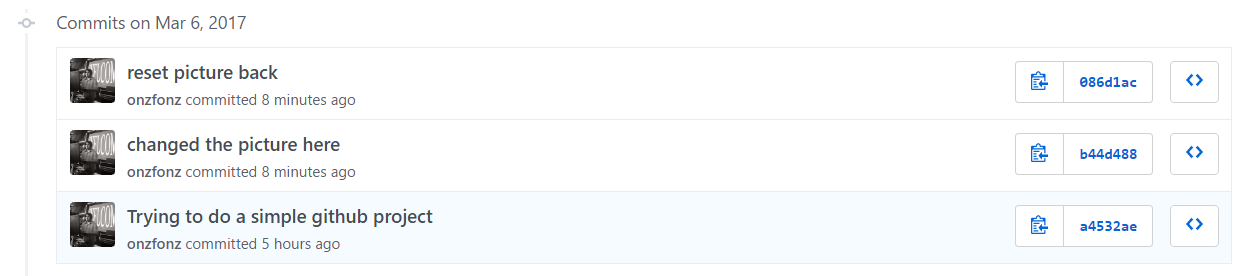
Once you have committed, go ahead and push the files to the server. Type **git push** and read the messages that are given to make sure that the push was indeed successful.

## Step-11

Once you have that finished and you pushed, then I will know that you have completed the tutorial, since your changes were pushed to the server you can simply check on github.com and click on the repository file to see the changes that were made. If github.com has the change you pushed that’s part of the credit. The other part of the credit will come from the settings. In github.com you can go to your repository and click on the commits text.



When you do that, you’ll see a list of the changes that you’ve made to the project. You should see a commit that has your picture and name on it.



Click on the text that you made for your particular one to see the changes that you committed and how your commit changed the file, which you can immediately identify. If you have this, then you’ll get credit if you have something like this show up in your repository along with your picture and id. No need to upload anything as all your changes will be stored on github. Hooray!



That’s it for now! We’ll do more with git in the future, but this is just meant to get you ready to work on the git lab with your team on Wednesday.