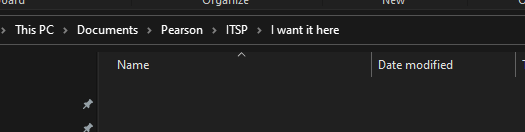
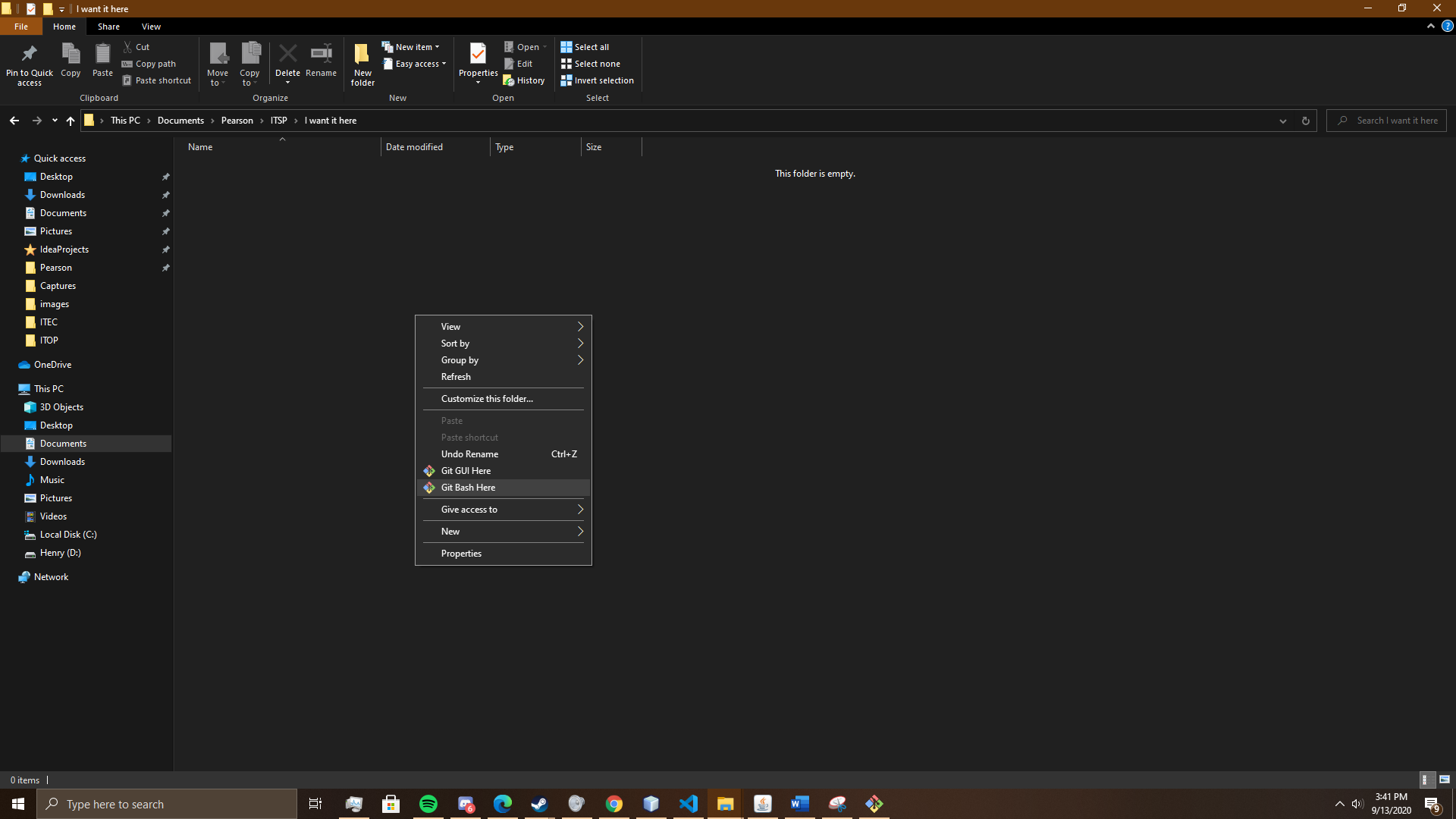
Once you created an account and have been added to the repo:

On your file explorer, navigate to the place where you want the project:

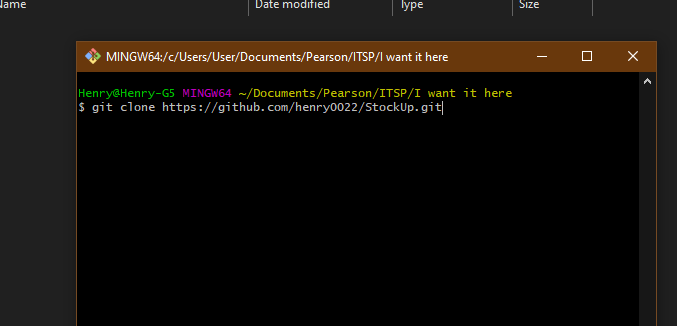


Open your console in this location. I use GitBash (cool console that you can download):

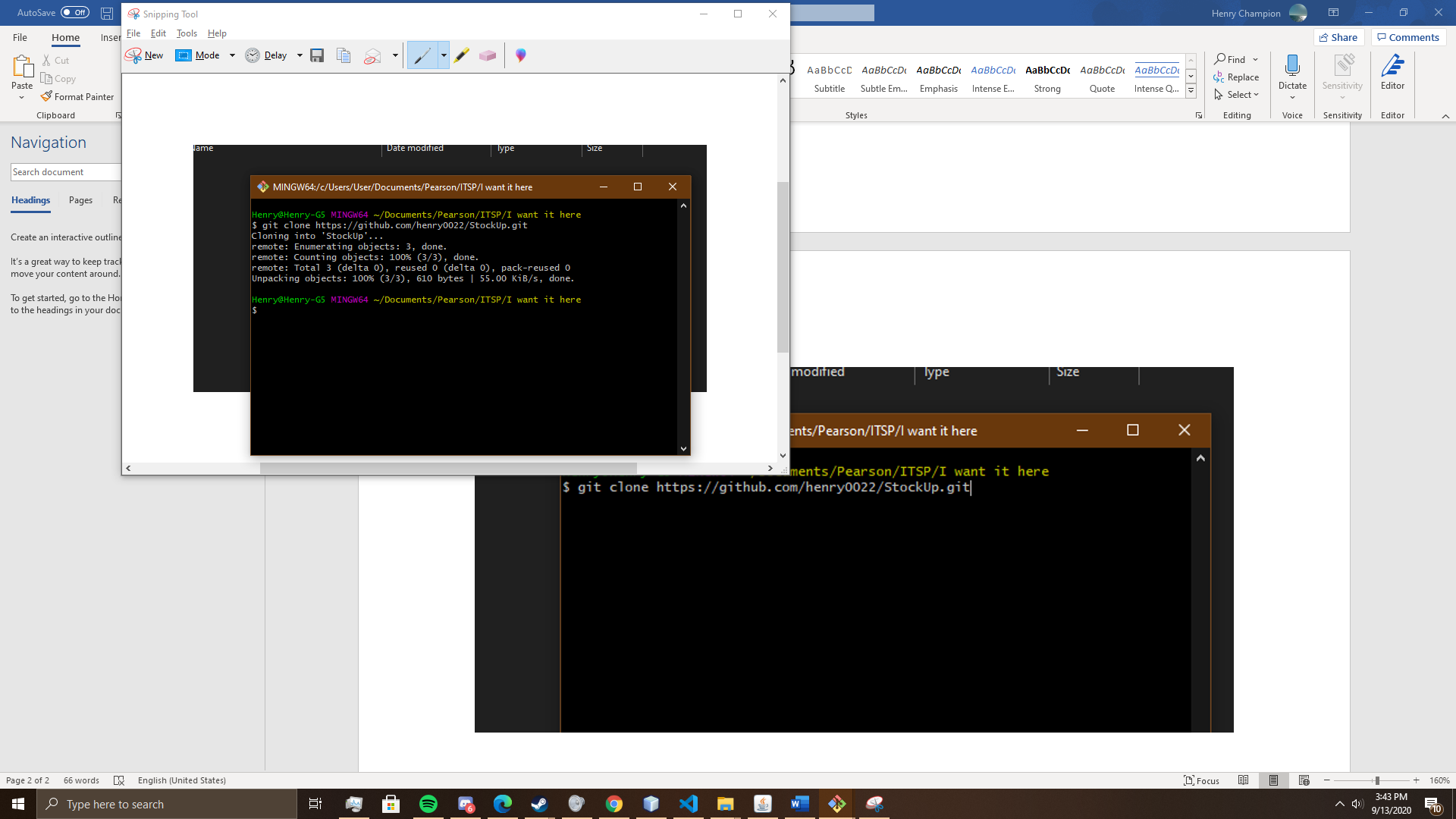


Type in : git clone <https://github.com/henry0022/StockUp.git>

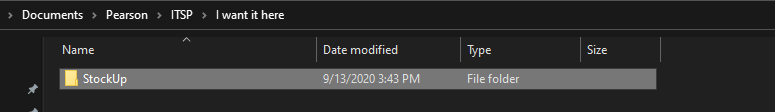
Hit enter and it should (after maybe asking for your github account credentials) copy everything from the repo.



Hit Enter:

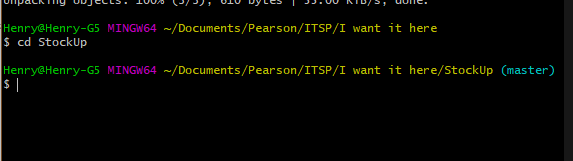


So now you should see the folder you cloned in this directory:



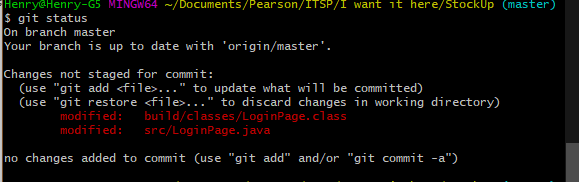
To ensure you are checked into the branch, and that you’ll be able to use git to work in this folder, you can do the following:

Navigate into this ‘StockUp’ folder using the console. Next to the folder name, it should now display ‘(master)’:



Now, any changes you make to the contents of this folder will be tracked by git.

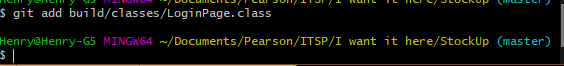
To view the unapplied changes through the console, type in ‘git status’:



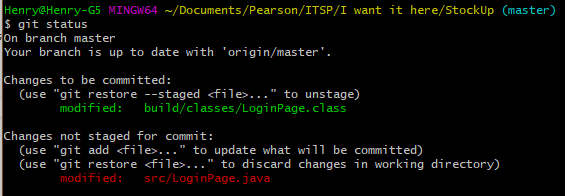
If you are happy with the changes, and want them to go to the repository, do the following:

For each file you want to commit, type in “git add exampleFile.extension”

In the above scenario it would look like this:

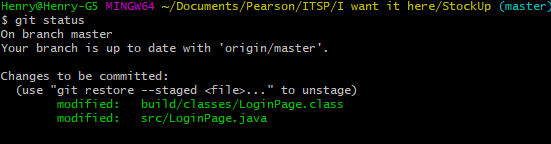


Now, when you run ‘git status’ again, the console will indicate that the file is now staged (ready) for a commit:



Alternatively, if you do not want to add each and every file independently, you can use the command “git add .” This will add and stage everything for the commit:



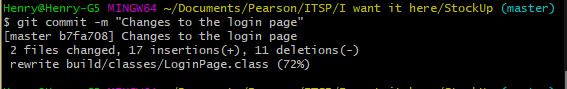


Now that everything is ready for the commit, you will give the commit command, followed by a message explaining what your intentions were for the changes you have made.

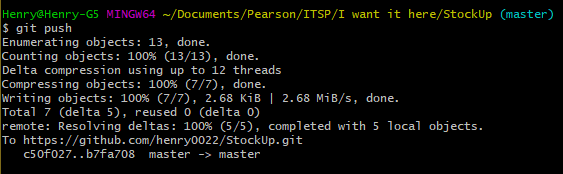
Command:

**git commit -m “Example message”**

Example for scenario:



Now you can proceed with ‘git push’. This will finalize the process and send the code changes to the repository so that other people can access it:



**NB:**

Whenever people have made changes, you use the command “git pull” to merge all of the changes into your ‘version’.

However, there is something everyone should try to avoid. This is called a **merge conflict.** This is caused when two guys were working on the same file, on the same section of code. Now they both made changes to that same section, but their solutions are different. The first guy that pushes his code will then cause that when the second guy attempts to push his, the process will fail with a message indicating the occurrence of a merge conflict. Think of it like a sort of deadlock.

These can be solved by the branch manager though, but it’s quite a tedious process as they have to manually select which one of the two commits / parts of the commits to actually merge with the master branch of the project.

**To avoid this from happening, try to always work on different sections of code simultaneously, or even better, on different files.**