

# Introduction

Before we begin using git version control and github, we must learn what both of these things are. Simply put, git is the most popular and downloaded version control software in the world. Version control software lets users keep track of all changes and updates to a project, and lets multiple people contribute to the same project at one time (sort of similar to google docs). The folder where a git project is stored is known as a repository.

Because code needs to be internally consistent to run, you can't have everyone editing the same document at one time. Instead every group member has a local copy of the repository which is linked to the main copy (remote repository) stored somewhere else (github.com in our case). When a person is done with a task or distinct item of work, they can 'push' their changes to the remote repository from which other project members can 'pull' the newest version so that everyone is up to date.

This is the heart of version control, and it occurs in three distinct steps. The terminology changes between different software, but the core principles are the same. Once a user is happy with a file or update, they add it to the index, or staging area. This tells git what to look at for changes. Once they are done with all of the current set of edits, they commit the changes. They also include a short commit message so that they will know what was added and or removed when they look back. This is a vital step where a lot of people slack off. Coders have been fired over repeated bad commits, it really is no joke. The message also lets other people know what was changed, not just the original coder. The last step is pushing the commit. This step uploads the changes to the remote repository for other people to download.

But why so many steps? Wouldn't it just be easier to cut out the commit and push automatically? Well not exactly, the commit adds a crucial point where the coder communicates with the other group members and decides if the project is ready for their changes. Sometimes it isn't because

other people are working on the same file, or other dependencies haven't been finished. If your code depends on someone else's, you want to hold off and wait.

In order to receive the changes that someone else has committed, you fetch and pull them. Fetch checks with the repository to see if anything has been done, and pull adds them to your local repo. Sometimes pull does both parts, but it is important to use it because git occasionally misses these changes if not prompted.

Here are the steps of git:

1. Add files that will/have been changed to the index
2. Commit changes (including a commit message)
3. Push changes to the remote repository
4. Fetch/Pull changes from the remote repository

Git usually lets the project integrate two versions of a file as long as the new edits don't interfere with each other. But what happens when the edits would interfere? What happens when someone needs to write some code that would temporarily break the project? These situations are where branching comes into play. Branching lets the users split a project into parallel versions, while maintaining the option to merge them back together when everyone has finished their work. As a result large projects have a main branch that is the most up to date version of the code, and working branches for different tasks. Thus each person won't get errors if the other branches are not ready. More information on the specifics of branching will be given later on in this tutorial, as the implementation changes based on the software used.

Additionally version control allows access to every step of a project, meaning that the manager and or project members can have a better understanding of the details of a project, and a holistic view. The most up to date version of a project (known as the branch head) can also be reverted to any previous version which allows for easy reversal of any mistakes or changes at any step.

# Install Git

The first step is to install git, the program that actually communicates between your computer and the remote repository.

1. Windows
  - a. Go to <http://git-scm.com/download/win>. The download will start automatically.  
Follow the instructions in the download wizard. For our purposes you should use the default setting for everything.
2. Mac
  - a. Open the terminal
  - b. Run the command: `$ git --version`

## Remote Repository

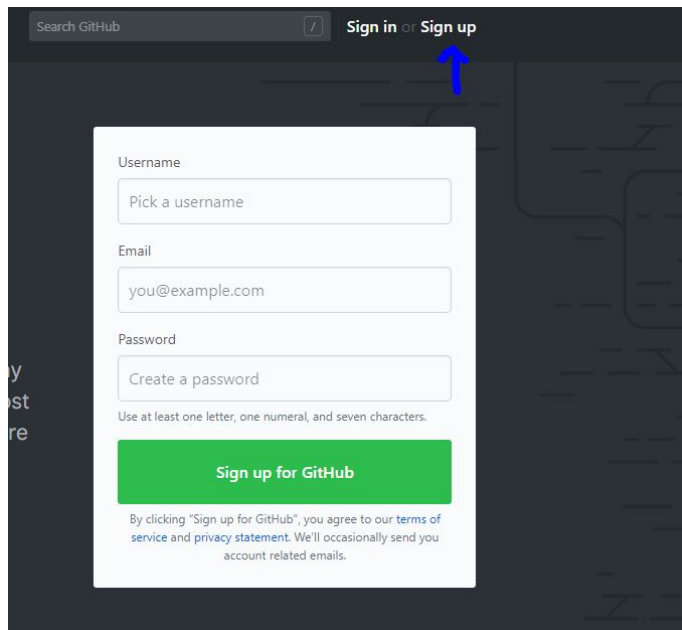
While there are many sites for hosting your remote repository, we will use github. On github all repos by free accounts are public which means that anyone can access them and look at your code. While this is a disadvantage for companies that want to protect their intellectual property, it works well for us.

## Github.com

Creating your account:

1. Go to [github.com](https://github.com)

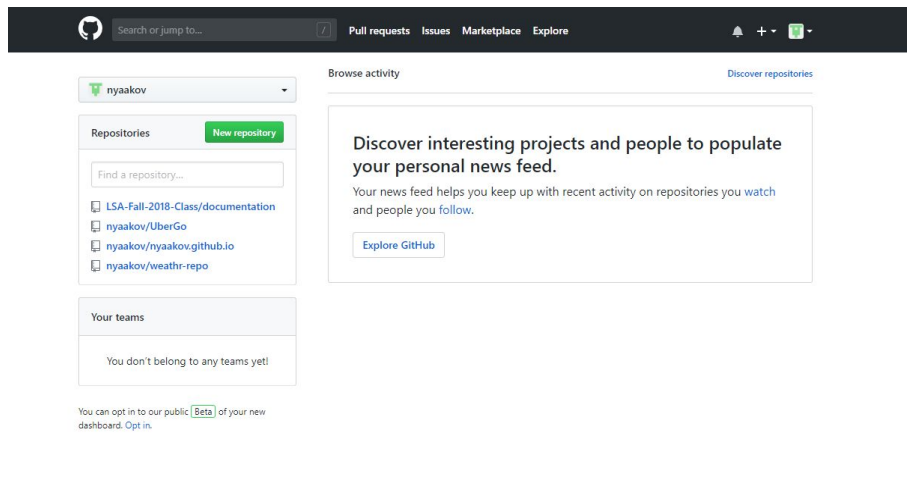
2. Select the sign up option



The image shows the GitHub sign-up page. At the top, there is a search bar and a navigation bar with "Sign in" and "Sign up" links. A blue arrow points to the "Sign up" link. Below the navigation bar is a white sign-up form with the following fields: "Username" (placeholder: "Pick a username"), "Email" (placeholder: "you@example.com"), and "Password" (placeholder: "Create a password"). Below the password field is a note: "Use at least one letter, one numeral, and seven characters." At the bottom of the form is a green button labeled "Sign up for GitHub". Below the button is a disclaimer: "By clicking 'Sign up for GitHub', you agree to our [terms of service](#) and [privacy statement](#). We'll occasionally send you account related emails."

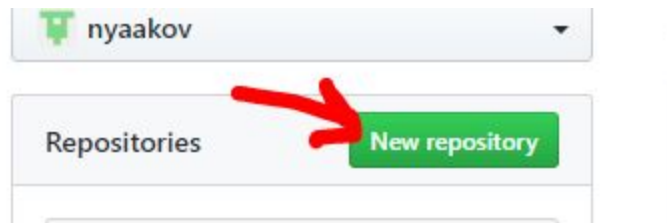
3. Once you sign up you will receive a verification email.

4. After you become a verified member, your user profile will look like this: On the left we can see a list of all of our active repositories, and read through them on github.



Creating a remote repository:

1. Press on the New repository button



2. Name your repository (can't have the same name as another repository that you own). Adding a short description can be useful for finding a project that you've already worked on. For now you don't have to worry about initializing with a readme, or adding a gitignore. As for the license press the button and type MIT. Select the option that shows up. (For more info visit <https://opensource.org/licenses/MIT>)

## Create a new repository

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

Owner



Repository name

Great repository names are short and memorable. Need inspiration? How about **laughing-garbanzo**.

Description (optional)



**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.

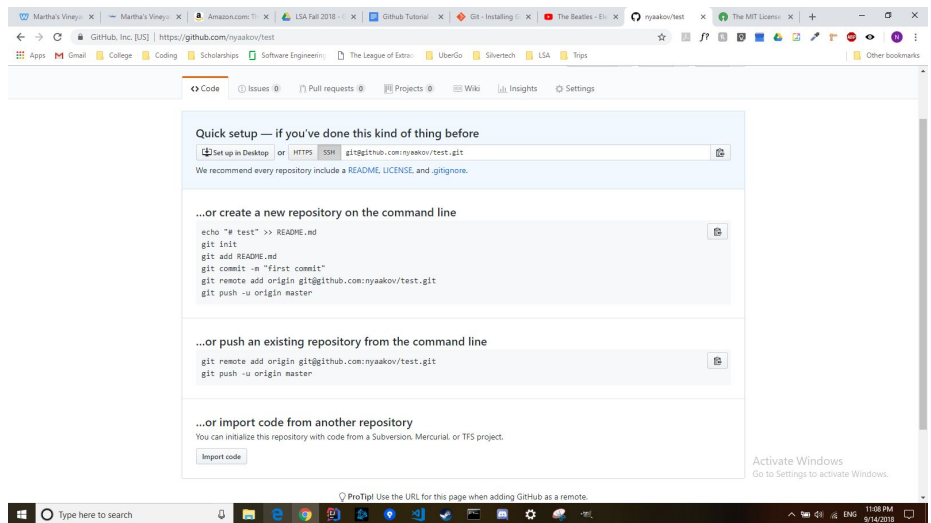
Add .gitignore: **None** ▼

Add a license: **None** ▼



Create repository

3. Once you have finished this step, your page should look like this:



4. For now the only part we care about is the link at the top:
5. This link is what we will use to connect our remote repository to the local one. We are done with github for now.

## Local Repository

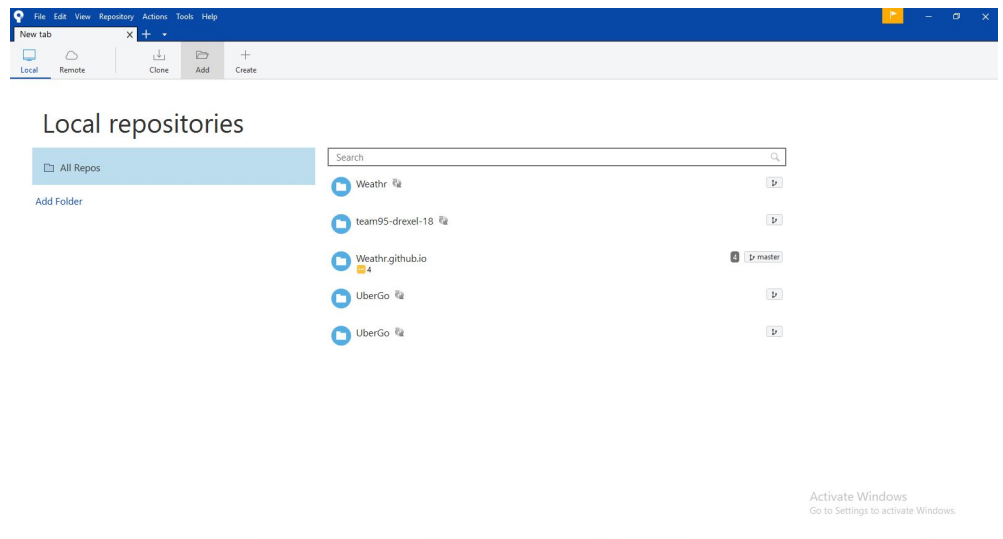
This is the local “copy” of the repo in which you add, remove, and edit content. For this tutorial we will be using sourcetree to visualize and interact with the repo. There are many ways to do this, but sourcetree is fairly intuitive and easy to use. Due to me having a windows computer, this tutorial will be in windows, the mac version is similar, but not identical. Although it works better with bitbucket, sourcetree also interacts with github.

## SourceTree

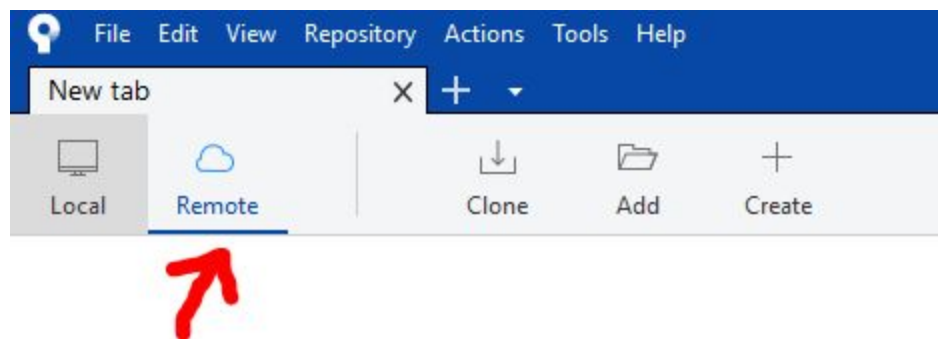
Download:

1. Visit <https://www.sourcetreeapp.com/> and choose your download version
2. Follow the steps in the install wizard. Standard setting are good for our purposes.

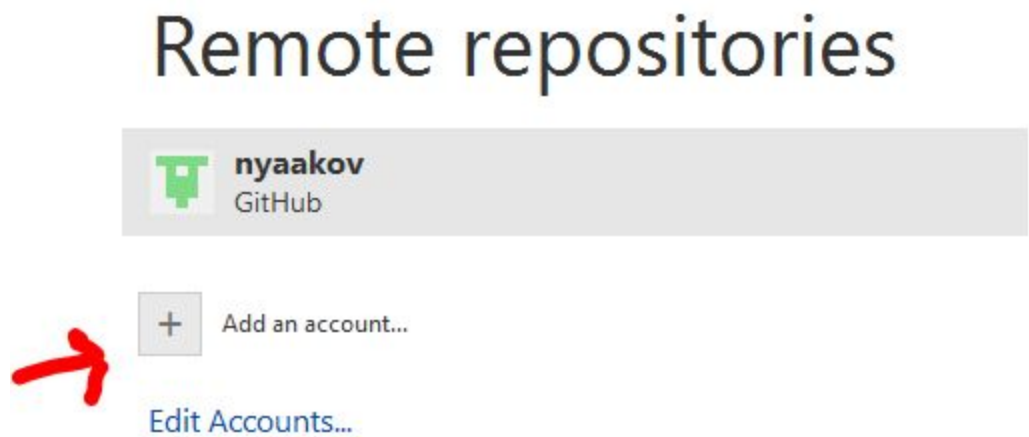
3. Once you've downloaded it, your screen should look something like this (minus the local repos to the right):



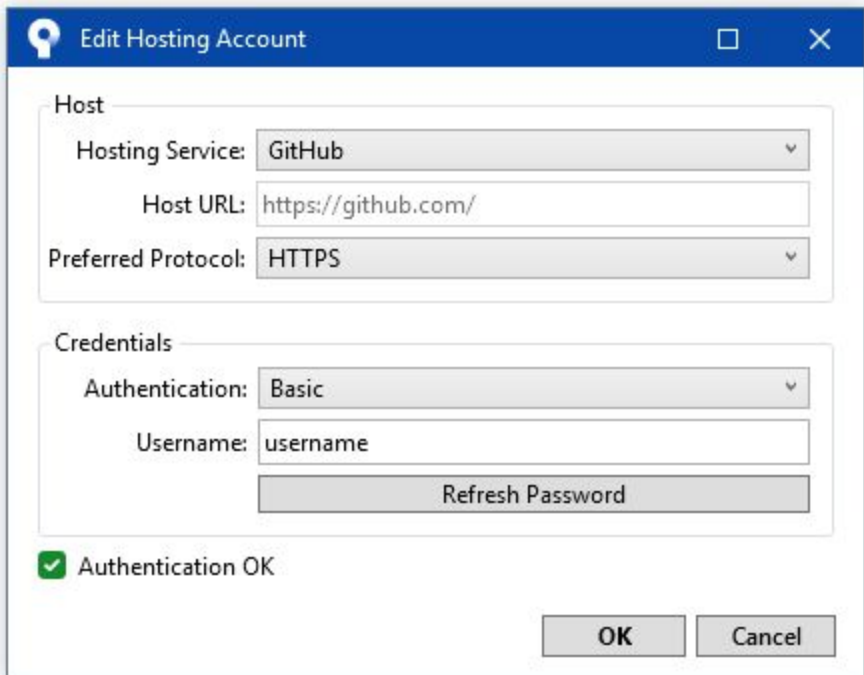
4. Select the remote button at the top left:



5. Select the add account button



6. In the menu that appears select github as your hosting service, and basic as your authentication (The preferred protocol doesn't matter very much) Input your github username. The first time you sign in you will probably also be prompted for your password:



The screenshot shows a dialog box titled "Edit Hosting Account". It has two main sections: "Host" and "Credentials". In the "Host" section, "Hosting Service" is a dropdown menu set to "GitHub", "Host URL" is a text field containing "https://github.com/", and "Preferred Protocol" is a dropdown menu set to "HTTPS". In the "Credentials" section, "Authentication" is a dropdown menu set to "Basic", "Username" is a text field containing "username", and there is a "Refresh Password" button. At the bottom left, there is a green checkmark and the text "Authentication OK". At the bottom right, there are "OK" and "Cancel" buttons.

7. Once you've signed in, all of the repos hosted on your github should appear. Find the one you want and press clone:



The screenshot shows a web interface for viewing repositories. At the top is a search bar. Below it is a checkbox labeled "Show Organization Repos" which is checked. To the right of this checkbox is a "Refresh" button. Below the checkbox is a list of repositories, each with a folder icon, the repository name, and the text "GitHub" below it. The repositories are: "nyaakov.github.io", "test", "UberGo", and "weathr-repo". To the right of each repository name is a "Clone" button. A red arrow points to the "Clone" button for the "test" repository.


8. The next page will look like this. All of the info is entered for you, all you need to do is press the clone button. You can also edit the



location of the repo on your computer by entering a different path on your computer. You can also browse to find one manually.

## Clone

Cloning is even easier if you set up a [remote account](#)

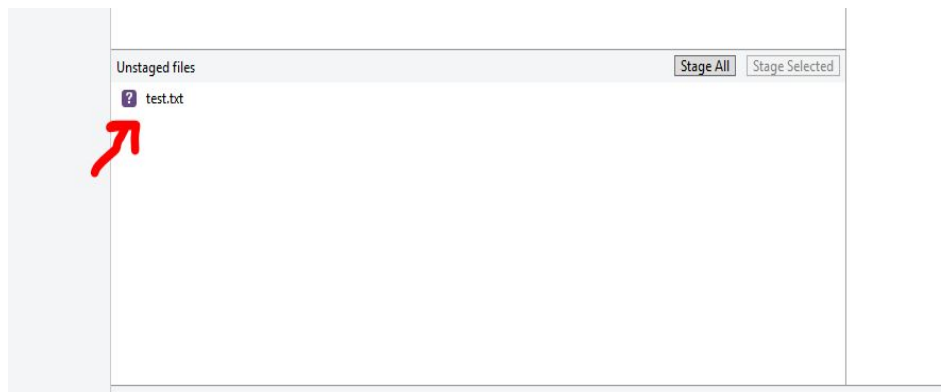
Repository Type:  This is a Git repository

Local Folder:

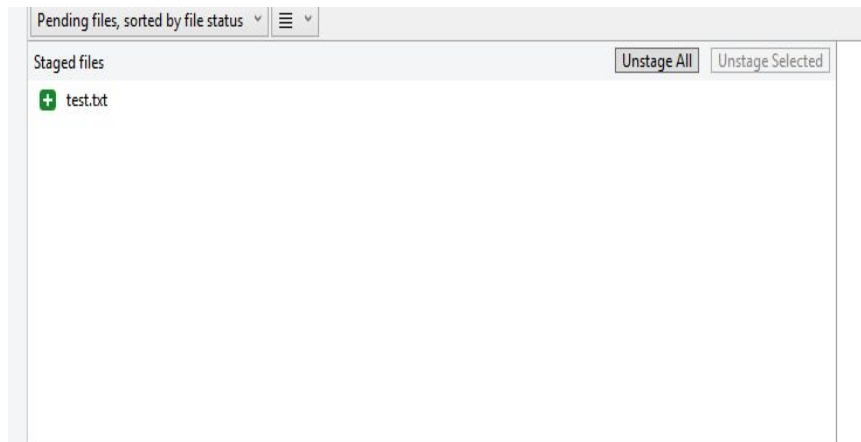
☒ Advanced Options

9. You can also do the above steps a slightly different way by selecting the clone option in the top left, and pasting in the link from the repositories github page. Both http and ssh options are available and it's only important that if you choose ssh you copy the ssh link. Both work fine.
10. You can also add repositories that are already downloaded or create new ones. But I won't go into details about this. The process is quite similar, except in reverse.
11. Once you have finished step 8, find the folder on your computer (wherever you decided it would be), and add a file or folder. When you check back on sourcetree it should appear in the unstaged

section:

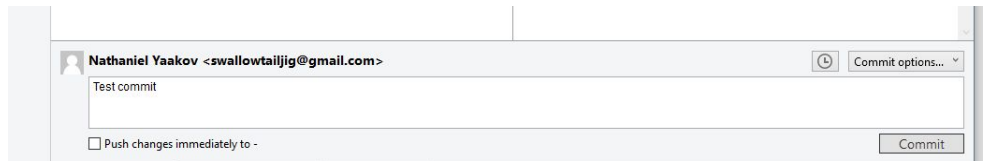


12. This section tracks all local changes to the repo, whether it be addition, deletions, or edits.
13. Once you are ready to stage, you hit the stage all, or select changes you like and hit the stage selected buttons. The changes should now appear in the staged files area:

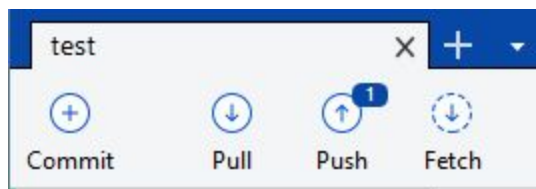


14. You can unstage a file or all of them at any time you like. The staging just adds them to the git change tracker.
15. Once you feel good about a set of changes (they are cohesive enough to be a group), you can commit them. The commit only grabs staged files, so make sure that the ones you want are in that section. As a recommendation, only commit code once it works to some degree. Always test before this phase.

16. The commit message is a vital part of this process. As I wrote earlier, people get fired over bad commits. The messages are useful for looking through the change log. Sometimes this is vital if the code breaks and you need to revert it.

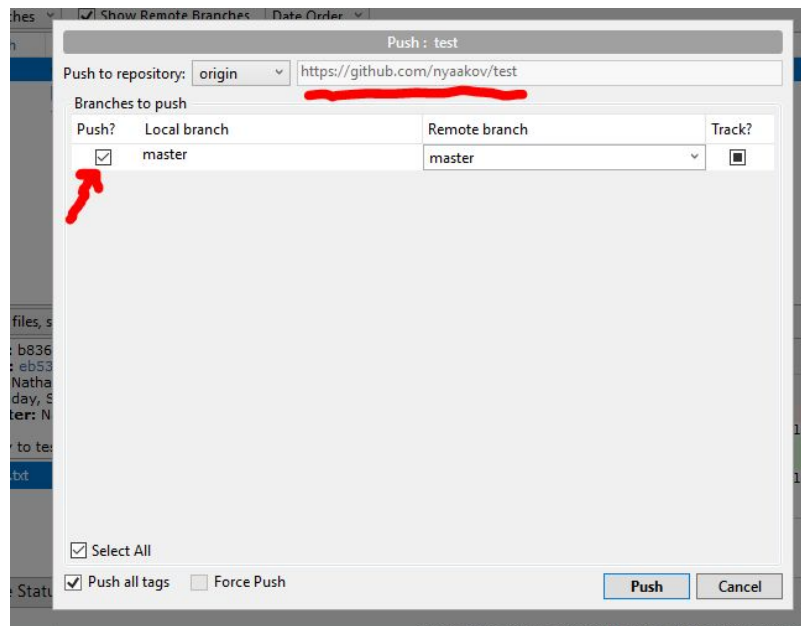


17. Once you hit commit, a small blue number should appear next to the push button. This means that you are ready to send your code to the remote repo. Only do this once you have communicated with your team and they are all up to date.



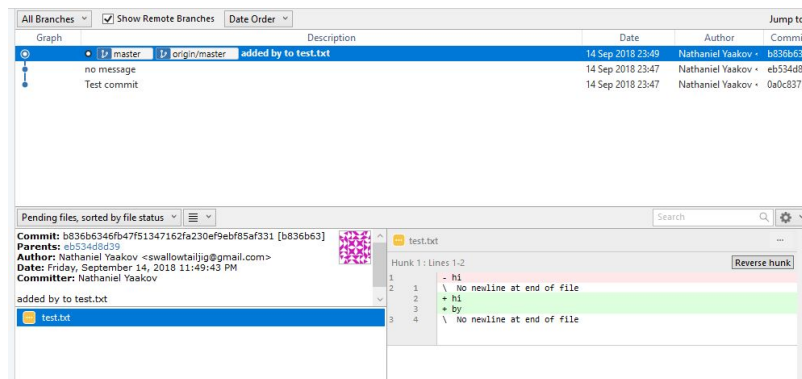
18. On the window that pops up make sure that the repository is origin, and that the link actually leads to your github repo (it should automatically). Select master in the branch option list, and press

push.



19. Lastly, I'll explain some of the other buttons next to the push one in the main options list. The commit button up top acts the same as the one down bellow. The pull button gets updates from the remote repo, and the fetch button checks if updates have been made. Pull usually checks as well, but ALWAYS press pull first because sometimes pull misses an update. We will ignore the rest of the buttons for now, as branching will be covered in the next tutorial. We will also go over doing this in terminal as understanding what drives the technology is important for more complicated actions, and resolving issues.
20. One last note, the log history and search tabs are extremely useful for reviewing your project. The log history gives a visual as well as written log of the commits to your project, and the search tab lets you look for commits based on the message (I told you it was

important!)



21. In log history, each node is a commit and the top one is what is referred to as the head (it is the current version). If you notice, it can be recognized by a white center instead of blue.
22. The final step is visiting your remote repository at github (github.com/username/reponame/commits/master and seeing the latest available version. Along with showing you the files, it tells you the latest commit, and you can view the commit history as well.

