

A Beginner's Tutorial to GitHub Repositories

Bryanna Hernandez

bhernandez2@oxy.edu

Occidental College

1 Introduction

GitHub is a valuable tool for computer science majors to securely store their files and projects online. However, its complexity often presents high barriers to entry for newcomers. This tutorial aims to dismantle these barriers through a step-by-step introduction to GitHub using Command Line. In this tutorial, beginners should expect to learn the basic commands to skillfully control the relationship between GitHub's four main areas: the Working Tree, the Staging Area, the Local Repository, and the Remote Repository. At the conclusion of this tutorial, beginners will also be given some tips on how to further master GitHub. This tutorial is accurate as of January 2024 and is geared toward the Windows Operating System.

2 Creating a Repository

In GitHub, a repository is a fancy word for a place to store your files and code. GitHub supports two different types of repositories. The first is known as a **Remote Repository**, which is hosted on GitHub's servers. This type of repository can be accessed from any local machine which means that if you were to lose your computer, for example, you would still be able to access your files.

Let's start by setting up a remote repository on GitHub's site:

1. Go to `https://github.com/` and create a profile if you haven't already.
2. Once logged in, you should be able to see your Dashboard. To the left of the screen there will be a section for "Top Repositories" and a green button called "New". Click this button.
3. You should see a screen like in Figure 1, asking you to fill in some data to initialize your repository. While these other fields have useful settings for later projects, for now, title your repository "Tutorial" and click "Create Repository".

Congratulations, you've created your first GitHub remote repository! Next up, we want to transfer this repository to your local machine, known as a **Local Repository**, through the process of **Cloning**:

1. Access your Tutorial repository's page on GitHub. From this screen, there will be a blue section, giving you the option to select between an HTTPS URL or an SSH key. Select HTTPS and copy the URL given.
2. Open your machine's command terminal and locate the folder you wish to initialize your local repository in.
 - For those unfamiliar with command terminal, you can change between folders using `cd <foldername>`.
3. Clone your repository by typing the command `git clone <httpslink>`.

Double-check that the cloning process worked by using your machine's file explorer. If you see a file called "Tutorial" then congratulations, you have successfully cloned your repository!

3 Adding Files to Repositories

Now that you have these two repositories, it's time to make use of them! A README text file is a great first file to add to a repository as it will introduce your project, so let's create one:

1. Open your machine's command terminal and open your Tutorial repository.
2. Create a README text file by typing the command `Notepad ReadMe.txt`. Your machine should automatically open the application.

3. Edit your README file and save it. For now, it's enough to type something simple like `hello world!` though in the future, you should aim for your READMEs to be more practical.

You've created your first file for your repository, however, it hasn't yet been uploaded to your local repository. Instead, it was created in what is referred to as a **Working Tree**. A working tree is an area where you can edit and create files that are useful for your project. However, in order to add files to your directory, you first have to transfer them to the **Staging Area**:

1. In your Tutorial repository, type the command `git status`. This will bring up all the files within your working tree, which for now, is only your README file. You will see that this file is considered untracked.
2. Add your file to the staging area by typing the command `git add README.txt`.
3. Now check your Tutorial's status once again. You should see that the ReadMe file is now considered a change to be committed. This means it has been successfully added to the staging area.

- If you ever want to take a file out of the staging area, use the command `git rm --cached <filename>`.

Once a file is in the staging area, there is one more step to add it to your local repository. While this may seem excessive, the staging area can be useful for projects with lots of moving parts. In the future, you can add multiple files to the staging area using the command `git add --all`. This will allow you to move all files to your repository in one step which may be necessary if a part of your project requires multiple files to be updated simultaneously to continue functioning and would break if only one file was uploaded.

As you may have seen in the last step, adding a file to the local repository is known as **committing** a file. Let's do that now:

1. In your Tutorial repository, move your file to the local repository with the command `git commit -m "adding ReadMe"`.
2. Check your git status. If your file was properly committed, you should see that your working tree is clean.
3. To check your commit history, use the command `git log`. Under a long alphanumeric string, you should see the date and time you pushed the commit, as well as your commit message.

Committing a file is often viewed as saving a repository, and when you make a lot of commits, it can be useful to differentiate these saves in your commit history by writing yourself a message, such as what was written above in the quotations. In the future, your commit message can be as long or short as you want as long as you include the quotation marks, but you'll thank yourself later when you're searching through your commit history if you keep them practical.

You've updated your local repository through committing, but updating a remote repository is known as **pushing** because it pushes your local files to your remote repository. However, you can only push whatever has been committed to your local repository; nothing in the working tree or the staging area can be pushed.

Let's push our README file to GitHub's remote repository:

1. In your Tutorial repository, type the command `git push origin main`.
 - As this is your first time pushing to the remote repository, you may be asked to login to your GitHub account in your browser.
2. In your browser, open your GitHub Tutorial repository. If you were successful, you should now see your README file. Because using a README file is standard practice, you should also be able to see the text within in the file right below, although other files can be accessed by clicking on their names.

The git push command has two important references. The first reference to origin is shorthand for the URL of the remote repository on your GitHub account. In most cases, just git push is enough, but if you wanted to push the files in your local repository to a different remote repository, then you would want to specify.

The second reference to main specifies which branch of your repository you are pushing to. In simple terms, a **Branch** is a tag associated with your commit that tells GitHub which version of your project to display to others. Your first commit is by default considered the main branch, but it may be useful in the future to have other branches dedicated to feature experimentation. When a branch is made, it uses the most recent commit of its parent branch, in this case main, as it's jumping off point. Branches allow you to make and save changes to your project without impacting the main version of your project. If you are only working with one branch, it is not necessary to specify in your push commands, but it is good practice!

4 Adding and Using Branches in Repositories

Let's make a new branch in your command terminal to demonstrate:

1. In your Tutorial repository, type the command `git branch side`.
2. To confirm this branch has been created, type the command `git branch`. You should see a list of branches, starting with `main` and then `side`. Additionally, the asterisk next to `main` verifies that we are currently working on this branch.
3. To switch branches, type the command `git switch side`.
4. Confirm with the command `git branch`. If you have successfully switched branches, the asterisk should be next to the `side` branch.

Branches can be named anything, `side` is just an example. It is best to name a branch something meaningful to you. Just as the `main` branch can be uploaded to the remote repository, so can a `side` branch. Now, let's edit the `side` branch further demonstrate how it works:

1. In your Tutorial repository, open your README file using the command `Notepad ReadMe.txt`. You'll notice that this is the same command we used to create the file.
2. Edit your README file by typing on the second line. Again, a simple sentence will do, but be sure not to edit the first line for now. Save your file.
3. Add this file to the staging area and commit the file to your local repository using the commands given above.
4. Now check your commit history. For now, only take note of the red tag attached to your commits.
 - Your commit history should look similar to Figure2.
5. Push your new changes to the remote repository.
6. Now check your commit history. This time, take note of the green and blue tags attached to your commits as well as the red tags.
 - Your commit history should now resemble Figure3.
7. Now switch your branch to the `main` branch and check your commit history for a third time.

The red tags on your commits reflect the last *pushed* commit of a branch. You should have noticed that on the first check, the `side` branch did not have a red tag because it had never been pushed. However, on the second check, it was given a red tag to reflect the updates made to the remote repository.

The green tags on your commits are the names of your branches, though it is important to note that these tags will only be attached to the most recent commit within that branch.

The blue tag that says "Head ->" indicates the last commit of the branch that you are *currently* working in. When you first checked your commit history, it would have pointed to the green `side` tag, and now it should be pointing to `main`. In fact, when switching your commit branch to `main`, you might have no longer seen your `side` branch commit history. It is good to note that branches have a hierarchy and will only show the commit log of its own branch and the parent branch(es) that were created before it.

Additionally, if you return to the tutorial repository on your GitHub website, you will see a button labeled `main` next to text that clarifies the number of branches pushed to your remote repository. If you click on the `main` button and select your `side` branch, you will be able to see the changes reflected to your README file.

5 Merging Branches

In the same way that you can create branches within your project, you can also combine your branches back into one. This process is known as **merging** and depending on the layout of your files, it may not succeed. First, let's demonstrate a successful merge:

1. In your Tutorial repository, ensure that you are on the `main` branch, this is the branch that will remain after the merger.
2. To merge branches, use the command `git merge side`.
 - A merged branch is automatically committed to the local repository.
3. Push these changes to your remote repository.

If you check your changes on the GitHub website, you should see that your README file in the main branch is exactly the same as the file in the side branch. This means it was successful. However, in a merge, the side branch doesn't disappear, it only pushed its changes to the main branch. You are still able to edit, stage, commit, and push files in the side branch without affecting the main branch.

Now let's attempt an unsuccessful merge, known as a **merge conflict**:

1. In your Tutorial repository, open the README file and edit the first line to say "hello world 1." Save the file.
2. Add this file to the staging area and commit the file.
3. Switch to your side branch.
4. Open the README file, and this time, edit the first line to say "hello world 2". Save your file.
5. Add this file to the staging area and commit the file.
6. Switch back to the main branch.
7. Use the command `git merge side` and attempt to merge the two branches.

This type of merging triggers a merge conflict and requires user action in order to complete the merge. A merge conflict happens when attempting to merge a file with competing changes on one line. In this example, the computer is unable to decide which number to keep after hello world and will not allow you to switch branches until the conflict is resolved. Let's attempt to resolve the conflict:

1. On the main branch, open the README file.
 - You should see a text file that looks similar to Figure4.
2. Edit the file so that it says "hello world 1" on the first line and is blank on the other lines. Save the file.
3. Stage and commit the file.
4. Push the file to the remote repository.

In this example solving a merge conflict is as simple as editing the text file. However, in more complicated merge conflicts, you may be required to edit multiple files to resolve the conflict.

6 Pulling and Stashing Changes

We have covered the basics of pushing local repository data to remote repositories, but there is also a method of transferring remote repository data to local repositories. This process is known as **pulling**, and can be extremely useful when working on a project with collaborators. Let's go through this process together:

1. On the GitHub website, in your remote repository under the main branch, click on the dropdown "Add file" button and choose "Create new file".
2. Title this new file "PullFile.txt", enter some file contents, and commit the file directly to the main branch.
 - When committing files on the GitHub website, staging is not required, and the extended description box allows you to write messages for the commit history.
3. Return to our Tutorial repository in the command terminal and use the command `git pull origin main`.
 - The references in this command work the same as in the pushing command.

You can verify files have been pulled by checking the commit history or by opening the file yourself. However, whenever changes are pulled from the remote repository, this can overwrite any work that is in your working tree or staging area. Therefore, it can be useful to temporarily save your changes before pulling to then restore your work later in a process known as **stashing**. Let's simulate this situation:

1. In your remote repository, create and commit a new file to the main branch titled "NewPull.txt".
2. In the command terminal, in the main branch, open, edit, and save your README file. Do NOT stage or commit this file.
3. Now stash your README file with the command `git stash`.
4. Use the command `git stash list`. You should see an item titled `stash@{0}`. This means you have properly stashed your files.

5. Reopen your README file and verify that the edits you made are gone.
6. Use the pull command to download your NewPull.txt file.
7. Use the command `git stash pop` to restore the edits to your README file.
8. Open your README file and verify that the edits have returned.

In this situation, you were able to download the NewPull file without losing your edits in your README file. Best practices advise you to only have one stashed edit at a time and to only use it in temporary situations. Other reasons entail wanting to stash work on one branch to go work on another branch for a change of pace, or to reduce the possibility for merge conflicts. Typically, it is simpler to commit files, but it is up to your judgement to decide when it is necessary to stash files.


7 Final Words

GitHub's complexity is somewhat infamous in the tech community. Therefore, while I hope that this tutorial has been a useful jumping off point, there are other great sources of documentation such as on <https://git-scm.com/> and the forums on <https://stackoverflow.com/> which have answered thousands of beginners' questions regarding GitHub. I encourage all users of this tutorial to seek out online resources whenever they encounter confusion, and above all else, happy coding!

Create a new repository



A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *  bry693 /

Great repository names are short and memorable. Need inspiration? How about [sturdy-succotash](#) ?

Description (optional)

- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

- ☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **None**

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

 You are creating a public repository in your personal account.

Create repository

Figure 1: The repository initialization screen, as an image.

```
C:\Users\19728\juniorcs\tutorial>git log
commit 8a9f42979796fa253fab55fd0a9724812b40d4ca (HEAD -> side)
Author: Bryanna
Date: Thu Feb 1 10:39:55 2024 -0800

    hello darkness

commit fc53c8b867428e989b0ccacc83ea4541cbe33a77 (origin/side)
Author: Bryanna
Date: Thu Feb 1 10:34:32 2024 -0800

    readme updated

commit b4d0c73b32e9c7d80416338d2f7655550e4a6 (origin/main, main)
Author: Bryanna
Date: Thu Feb 1 09:15:06 2024 -0800

    readme
```

Figure 2: Commit history before pushing the side branch.

```
C:\Users\19728\juniorcs\tutorial>git log
commit 8a9f42979796fa253fab55fd0a9724812b40d4ca (HEAD -> side, origin/side)
Author: Bryanna
Date: Thu Feb 1 10:39:55 2024 -0800

    hello darkness

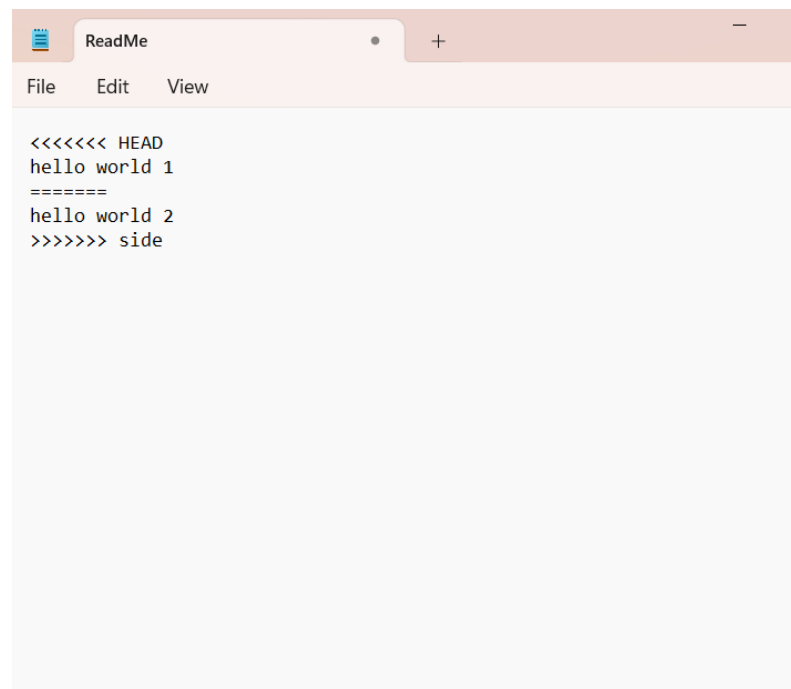
commit fc53c8b867428e989b0ccacc83ea4541cbe33a77
Author: Bryanna
Date: Thu Feb 1 10:34:32 2024 -0800

    readme updated

commit b4d0c73b3b32e9c7d80416338d2f7655550e4a6 (origin/main, main)
Author: Bryanna
Date: Thu Feb 1 09:15:06 2024 -0800

    readme
```

Figure 3: Commit history after pushing the side branch.



The screenshot shows a text editor window titled "ReadMe" with a menu bar containing "File", "Edit", and "View". The text content of the file is as follows:

```
<<<<<< HEAD
hello world 1
=====
hello world 2
>>>>>> side
```

Figure 4: README file experiencing a merge conflict.