Junior Seminar - GitHub Tutorial

Zahir Choudhry

The goal of this paper is to teach people about what Github is and how to utilize it, as well as issues you may face when using it. This tutorial will be utilizing Bash commands in Terminal (Mac), but there are alternaticve ways to use Github such as Gitkraken and Github Desktop.

Table of Contents

Basic Concepts:

Advanced Concepts:

---

4 Data Stores of Github

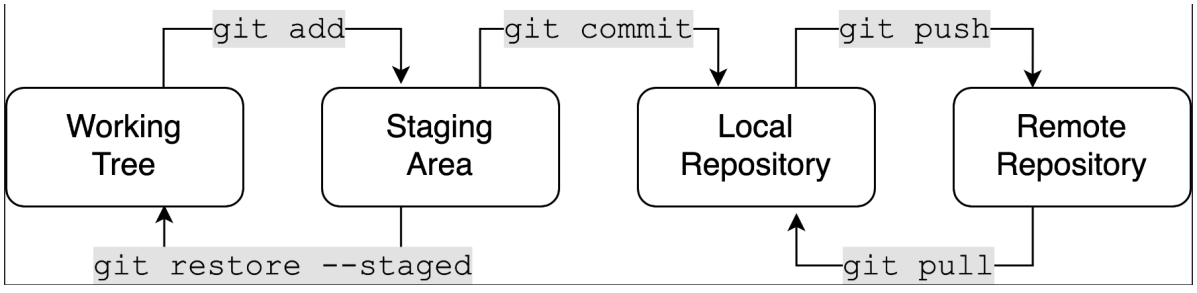


Figure 1: The 4 Data Stores of Git

Working Tree

The Working Tree is where the user makes changes to their files. It is the project directory on a user's local device. Basically, it is the files on a user's computer.

Staging Area

The Staging Area is the intermediate area where a file goes before it is committed to Github. Allows the user to check their changes and make sure there will be no conflict with other files that are also staged or going to be committed.

Local Repository

The Local Repository is a copy or clone of a user's Github Repository that resides on their computer. The Local Repository will contain the history of your Github project, including all branches, commits and version control information. Allows for users to work on their projects offline and collaborate with others.

Remote Repository

The Remote Repository is a repository hosted on another server from your own device. It serves as the version of your project that others can clone and work on, and is where you want to commit your most recent version of your code.

Basic Concepts

Init-ing / Cloning

Init-ing and Cloning are 2 commands that create new Github repositories on your local directory. However, there is a crucial difference between the two commands. Init-ing create a completely brand new blank repository in you directory, while git clone will pull a copy of an existing repository from a remote server onto your local device.

Adding

Adding is putting a file into the Staging Area. This area is still on your local device and is one level below committing your work. You must add a work before committing it. Adding is important because it allows developers to see what they may be committing that they may not want to, like conflicts or SSHs/Keys that they intended to delete.

Committing

Committing is the second step, moving from the Staging Area into the Local Repository. Once a file is committed, it is saved in the version control history, which is why the Staging Area is important, allowing users to look over their code one last time before it is permanently saved into Git's history. When a user makes a commit, a message editor pops up asking for a 'commit message'. This message allows users to provide detailed information on the changes they've made, making it easier for others trying to work on the code to understand how they can help and what has already been accomplished.

Pushing

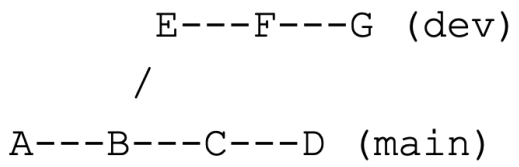
Finally, Pushing is moving the files from the Local Repository to the Remote Repository. Your project files are finally off your computer and on a separate server that other users can use to clone your project and work on it. It is a good practice to check that your Local Repository and Remote Repository are synced up using 'git fetch' or 'git pull'. When first pushing to a Remote Repository, the user has to specify both which Remote Repository and which Branch they intend to push to. When pushing to private server, users will be prompted with an authentication screen.

Advanced Concepts

Stashes

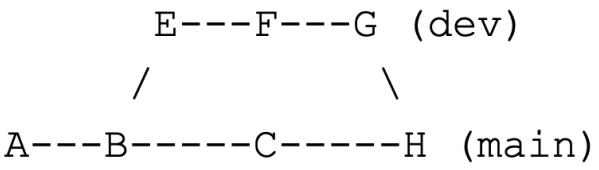
Stashing is a technique that allows users to temporarily not add certain changes in your Working Tree so that users can work on another task, and then come back and apply those changes when they are ready. Users can choose to apply changes from their stashes or delete them instead.

Branches



A Branch is like when a specific commit has 'multiple children'. Branches allow users to have parallel development, with one branch being used to work on one feature and another work on a different one for the same project, allowing for the two ideas to fully fleshed out before being combined into a final product.

Merging



Merging is when 2 branches are combined back together into one branch. While it seems simple, there are many opportunities for conflicts when merging. A conflict occurs when both files try to overwrite the same things, leaving the computer confused on what is the correct one to follow. When a conflict occurs, the merge is halted and marks files with conflicting information. The merge will resume once the user deals with the conflicts and re-instates the merge.

Examples of Conflicts

- Same line in both branches have different modifications
- File deleted in one branch, modified in another
- Directory names not the same
- Whitespace and Formatting
- Binary Files (Images/Compiled Binaries): Github can't perform a line-by-line merge on binary files. If both branches modify a binary file, Git marks it as a conflict

Applying What We Have Learned

Creating a Repository

Now that we have covered all the different concepts when it comes to using Github, lets make our own repository and practice using these techniques and ideas. Lets start by creating out repository, called ***Git-Tutorial*** in our Desktop. ***cd "Directory"*** means ***Change Directory*** and allows us to choose which directory we are in.

```
(base) Zahir@ZBook-Pro ~ % cd Desktop
(base) Zahir@ZBook-Pro Desktop % git init Git-Tutorial
Initialized empty Git repository in /Users/Zahir/Desktop/Git-Tutorial/.git/
```

Creating a File in our Repository

Now that we have made out lets make a Python File called ***Test.py*** to put in our Repository. We use ***touch "File-Name"*** to create the file and then use ***ls -la*** to list all files found inside of our ***Git-Tutorial*** directory. After running ***touch*** and ***ls -la***, your terminal should look something like this:

```
(base) Zahir@ZBook-Pro Git-Tutorial % touch Test.py
(base) Zahir@ZBook-Pro Git-Tutorial % ls -la
total 40
drwxr-xr-x  6 Zahir  staff    192 Mar 22 01:23 .
drwx------+ 9 Zahir  staff    288 Mar 22 00:42 ..
-rw-r--r--@ 1 Zahir  staff   6148 Mar 22 01:22 .DS_Store
-rw-------  1 Zahir  staff  12288 Mar 22 01:22 .NetrwMessage.swp
drwxr-xr-x  9 Zahir  staff    288 Mar 22 00:42 .git
-rw-r--r--  1 Zahir  staff      0 Mar 22 01:23 Test.py
```

Editing Files in our Repository

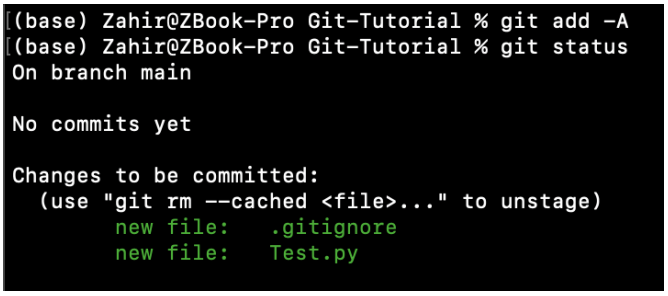
Now that we have created a python file, we can edit it and get it ready to stage and eventually commit it to our Remote Repository. While it is possible to use an IDE, for the purposes of building familiarity with using terminal, I will cover how we can use ***vim***. If we use the command: ***vim "File-Name"***, a window should pop. press ***i*** to start typing code. Once you are done writing code, press the ***escape key*** and type ***:wq*** [short for 'write and quit'] to save your changes and exit the text editor. This is what the text editor should look like:

You can write any text you want, but be sure to press the ***escape key*** and type ***:wq*** when you are done. You can open your file in a text editor or IDE to see that the file is now edited!

Adding / Committing Files in our Repository

Now that our file is edited, it is time to commit the changes to Git. Lets start by using ***git status*** to check our untracked files.

Here we have 2 untracked files: ***Test.py*** and ***.gitignore***[we can ignore this one].To track our files and move them into the staging area, we want to use ***git add "File-Name"*** to add a specific file or ***git add -A*** to add all untracked files in that directory. Here is what ***git status*** looks like after adding the files:



Now that the files have been moved to the staging area, we are ready to commit them using the ***git commit -m "Commit Message"***. The commit message should describe the changes you are committing in some way. In my example, I write "First Commit". This is what the results should look like:

```
[main (root-commit) 9ceb11a] First Commit
 2 files changed, 3 insertions(+)
 create mode 100644 .gitignore
 create mode 100755 Test.py
```

Branching

Congratulations! you have successfully committed a file to Github. Now we are going to look at creating a branch for our repository. Let us start by cloning our repository into a remote one. We first need to make a repository, called ***Git-Command-Branch***. After creating the repository, clone our tutorial repository into it. Use ***ls -la*** afterwards to make sure the contents match.

```
(base) Zahir@ZBook-Pro Desktop % git clone https://github.com/BiggestZ/Git-Command-Branch
Cloning into 'Git-Command-Branch'...
warning: You appear to have cloned an empty repository.
(base) Zahir@ZBook-Pro Desktop % cd Git-Command-Branch
(base) Zahir@ZBook-Pro Git-Command-Branch % ls -a
.       ..      .git
(base) Zahir@ZBook-Pro Git-Command-Branch % rm - rf .git
rm: -: No such file or directory
rm: rf: No such file or directory
rm: .git: is a directory
(base) Zahir@ZBook-Pro Git-Command-Branch % rm -rf .git
(base) Zahir@ZBook-Pro Git-Command-Branch % git clone ./Git-Tutorial
fatal: repository './Git-Tutorial' does not exist
(base) Zahir@ZBook-Pro Git-Command-Branch % ls
(base) Zahir@ZBook-Pro Git-Command-Branch % git clone ../Git-Tutorial
Cloning into 'Git-Tutorial'...
done.
(base) Zahir@ZBook-Pro Git-Command-Branch % ls -la
total 16
drwxr-xr-x   4 Zahir  staff   128 Mar 22 15:13 .
drwx------+ 11 Zahir  staff   352 Mar 22 15:11 ..
-rw-r--r--@  1 Zahir  staff  6148 Mar 22 15:13 .DS_Store
drwxr-xr-x   6 Zahir  staff   192 Mar 22 15:13 Git-Tutorial
(base) Zahir@ZBook-Pro Git-Command-Branch % ls -la
total 32
drwxr-xr-x   6 Zahir  staff   192 Mar 22 15:14 .
drwx------+ 11 Zahir  staff   352 Mar 22 15:11 ..
-rw-r--r--@  1 Zahir  staff  6148 Mar 22 15:14 .DS_Store
drwxr-xr-x  13 Zahir  staff   416 Mar 22 15:07 .git
-rw-r--r--@  1 Zahir  staff    46 Mar 22 14:17 .gitignore
-rwxr-xr-x@  1 Zahir  staff    60 Mar 22 01:37 Test.py
(base) Zahir@ZBook-Pro Git-Command-Branch % 
```

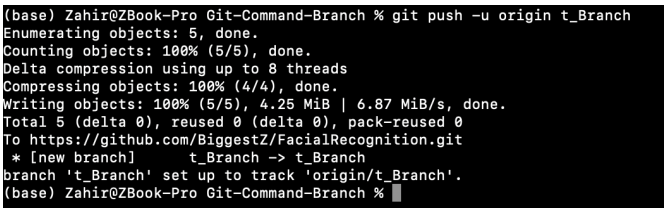
Now that our repository is copied, let us use ***vim*** to create an addition to out ***Test.py***.

```
print("Hello! This is a test file for our new Repository!")
print("Now we are branching our Repository")
~
~
~
~
~
~
```

Now before we push our new ***Test.py*** to a remote repository, we need to create a temporary branch in the current repository. We want to make sure we are in ***Git- Command-Branch*** and use the command ***git checkout -b [name]***. To check which branch we are on, use ***git branch***. The current branch will be highlighted green with a start next to it. We have successfully created a new branch! In case you ever wanted to delete a branch, use ***git branch -d "branch name"*** and be sure to push that after running.

Merging

To Merge the branch back to the main branch, we first want to push out branch using ***git push -u origin "branch name"***



After doing so, we want to move back to thee main branch with ***git checkout main*** and then use ***git merge branch name*** to merge them together.

Stashes

Using ***git stash*** allows us to save our work so that we can reapply it later. After stashing, your work tree should be clean and all changes should be saved, something that can be checked when using ***git status***. To bring stashed work, use ***git stash apply***. All work will return when using ***git status***.