Assignees

Now that our issues are correctly labeled, it is time to assign them to a developer. It's a fairly easy task , not so different from labels.

You can assign an issue to up to ten members of your team. However, since you're the only one right now, you can only assign yourself a task. Let's do it!

Navigate to the issues titled "Data preprocessing" and "Model selection", and assign them to yourself.

Assigning an issue to a team member works exactly like adding labels. You can check the Figure below for an example.

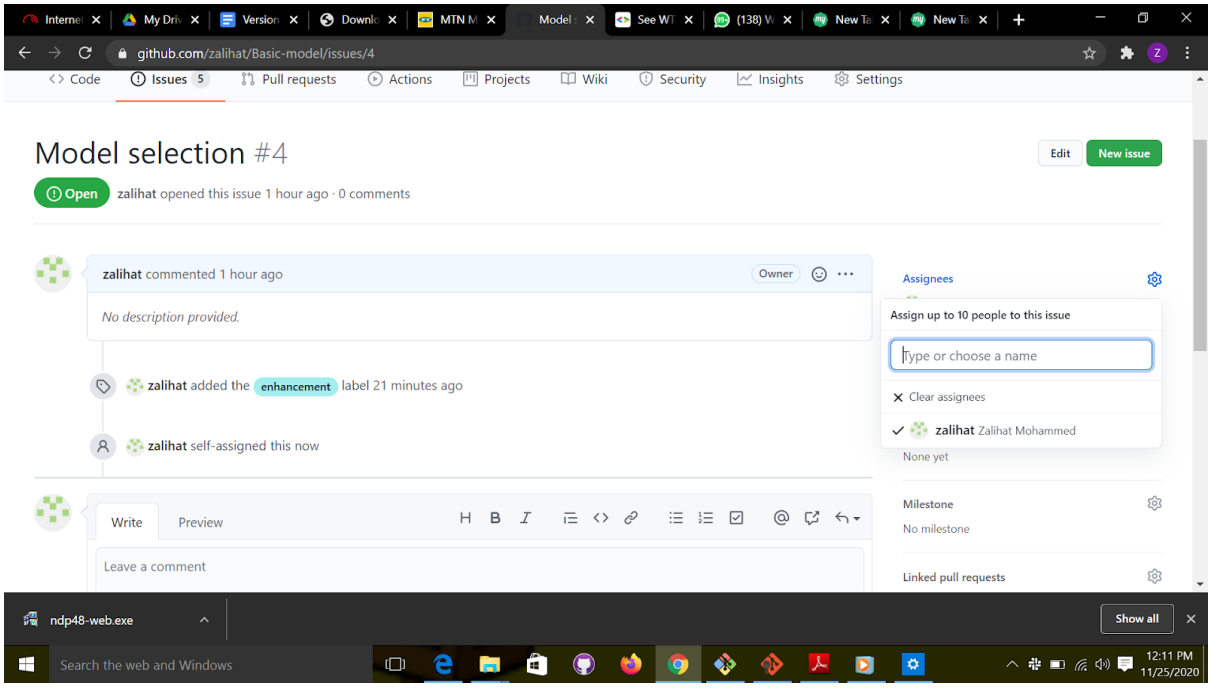

Figure 28: assigning people to an issue

After you assign those two issues to yourself, you will get a result like the one shown in Figure below on your Issues page. You can now filter through your issues by labels and assignees.

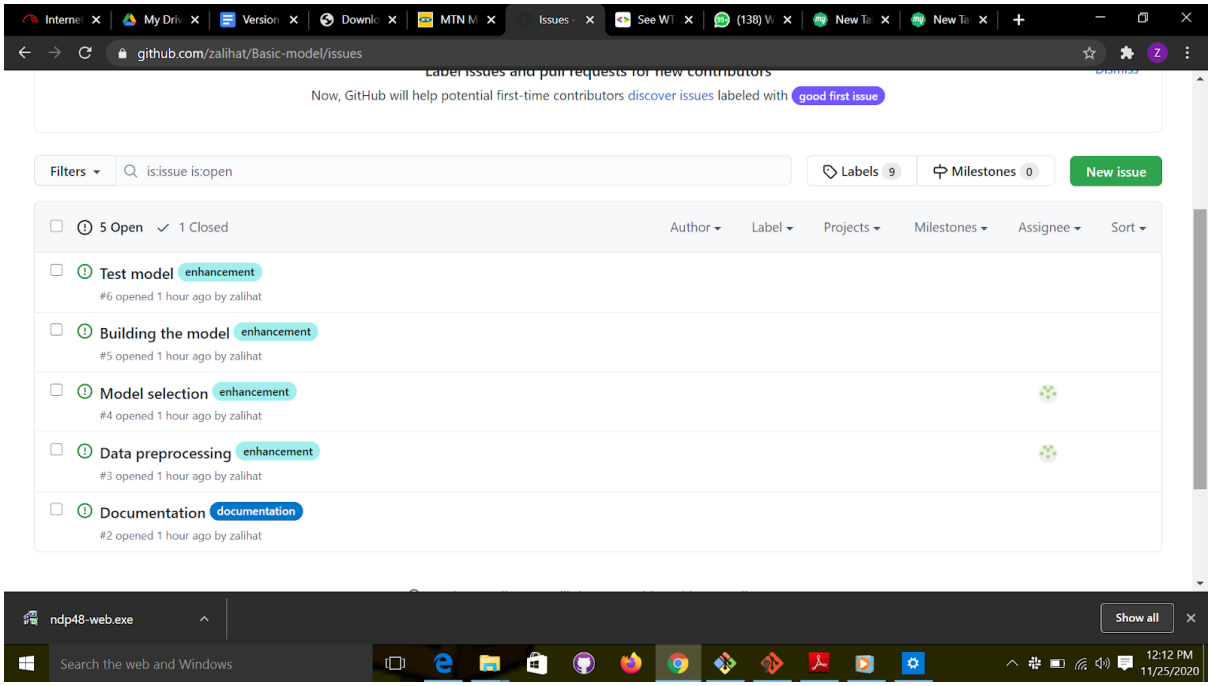

Figure 29: filter issues using labels

Git fork and git clone

The concept of “forking” and “cloning “is very important when we collaborate with other people to work on a project. Before you make changes to a repository, you need to have a copy of that repository saved, either locally or remotely. When you want to clone(copy) a project to your local machine, you use git clone and if you want to copy that repository to your own repository you use git fork.

Here are the guidelines for contributing to projects on github

1. Fork the repository, so you have a copy on your repository.
2. Clone the repository, so you have a copy on your local machine.
3. Make changes to it/ contribute to it.
4. Push it to your remote repository.
5. Send a Pull request to the owner of the repository.
6. If the owner of the repo is okay with your contribution then they will merge your changes with the original repository.

To "fork" a repository, login to your github account and search for the repository you want to contribute to. Then click on “fork” and it will ask which account you would want the repository to be copied to

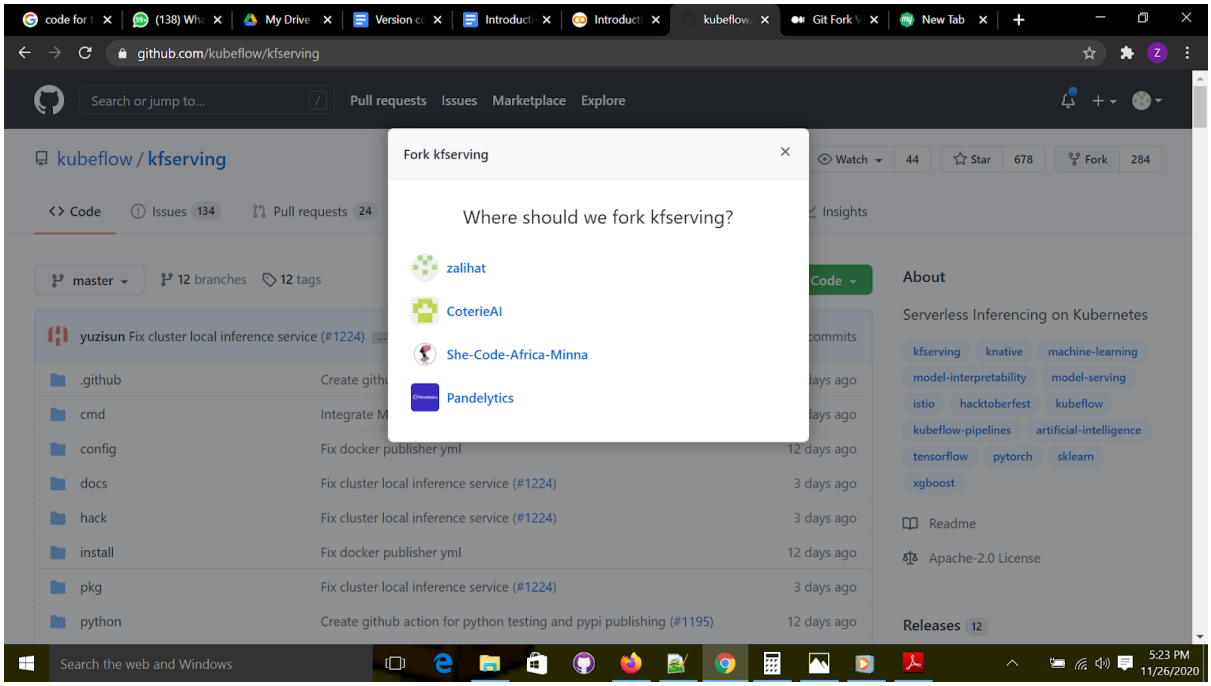

Figure 30: “forking” a repository

You will select the repository, which will lead you to something that looks like this,

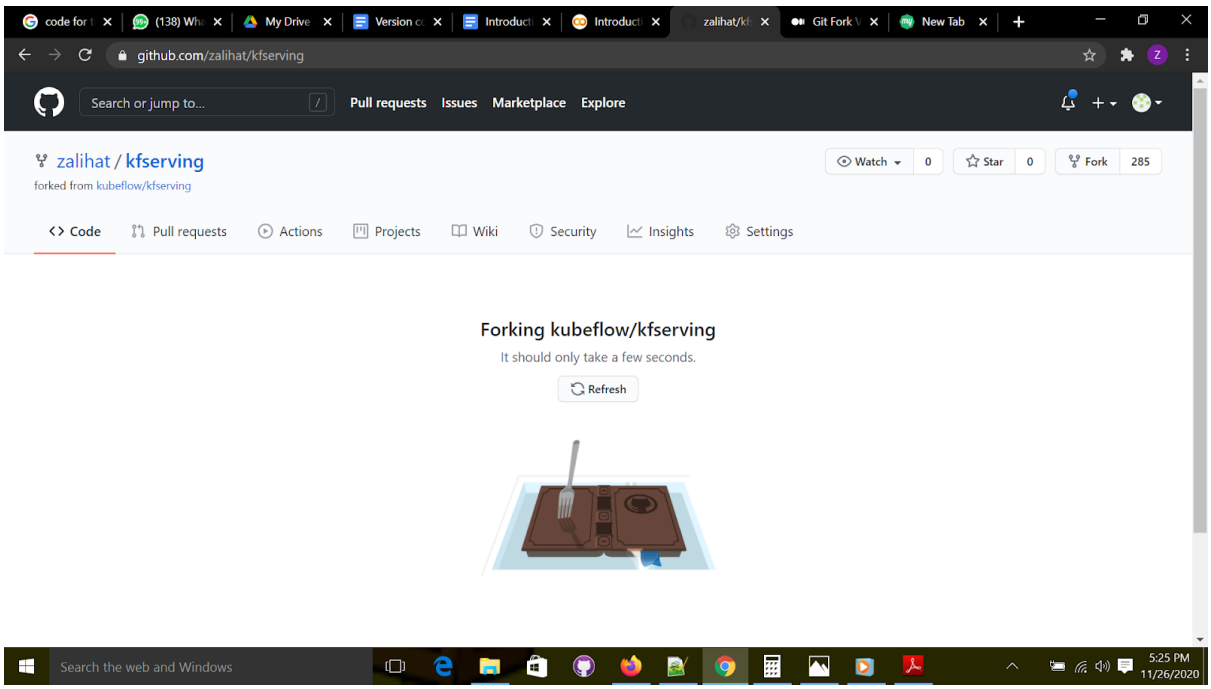

Figure 31: “forking” a repository

When this is done, you will find a copy of the repository in your account.

The second step is to copy the repository to your local machine. Here are the steps:

1. Go to your remote repository and copy the link as shown below;

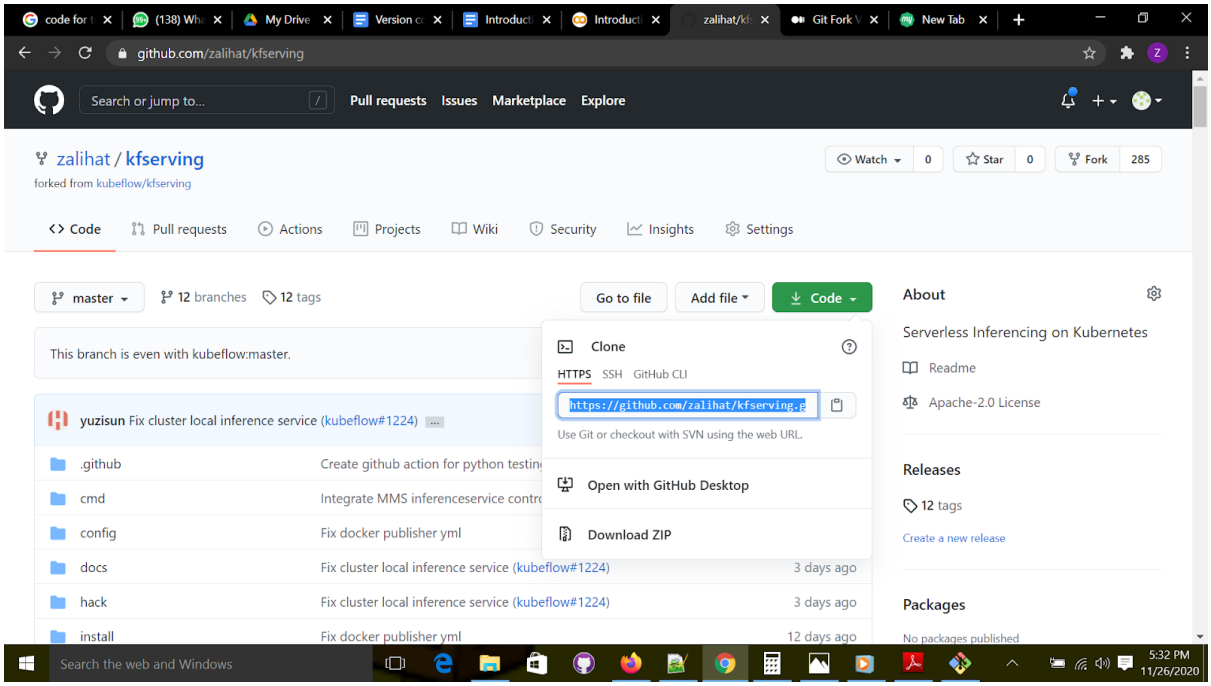

Figure 32: Copying the repository's url.

2. Go to the folder in which you want to clone the repository, and open "git bash". Run the command below to clone the repository to the folder

Syntax;

git clone url

Example:

git clone https://github.com/zalihat/kfserving.git

Note that to paste the url, use insert, copy the url to clipboard, and type git clone. Then click insert.

Now you have a copy of the repository that you can make changes to.

Branches

In this section, we will talk about the most common way that developers use GitHub.

Keep in mind that each team has its own way of doing things, but each of these ways of

working is inspired by the basic workflow that they are going to present.

Remember the little fact about making mistakes? This omnipresent possibility of mistakes is why you need to follow this GitHub workflow, such that even if mistakes occur, you can isolate their repercussions in a controlled manner. Our way of work from the previous chapter was to commit everything directly to the main project, which is very dangerous. The main project is most of the time the "production" line, the version that the clients see and use. Therefore, this version must be very clean and should always be exploitable. If any error makes its way to the main version, the clients will experience bugs and it will disrupt every team member's work.

One way to resolve this issue is to create a copy of the main project and work on this clone. Each change you make to this copy will not affect the main project, so none of your mistakes

can impact clients. Also, when you (and other people) are perfectly sure that the changes made resolve the issue, you can reproduce those changes in the main version.

How to create a branch

To create a branch, use the git checkout command. For example, let's say we are working on a model for predicting the weight of a baby. We do not want to make changes to the main branch till the changes are reviewed and approved by everyone on the team. Here is what we will do; Remember when we talked about github issues? We said you could create different tasks(issues) and assign them to members of the team. We create issues for each task, task may include, data collection, model training, etc. As a result of this, one thing we can also do is to create branches for each of those issues.

Syntax:

git branch branch-name

Example; we can create a branch for the data preprocessing by running this command;

git branch "data-preprocessing"

To check if the branch has been created, run the command;

git branch

This will give a list of all the branches on the repository, as shown in the figure below:

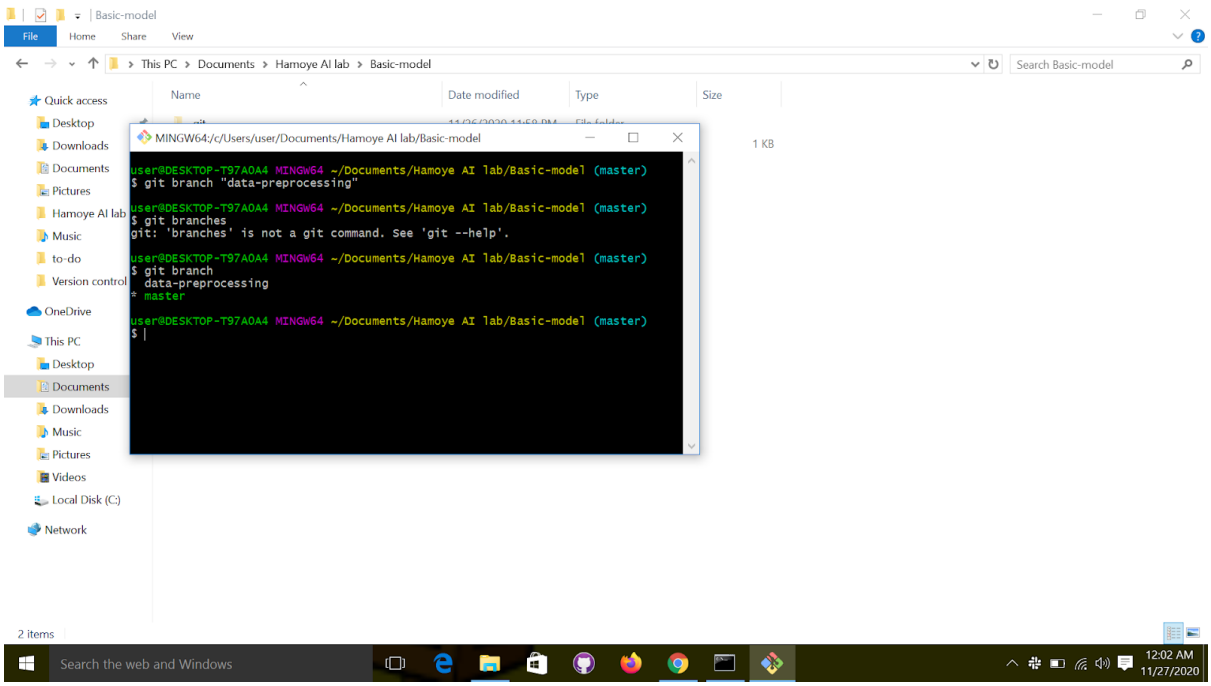

Figure 33: list of “branches”
Before you can make changes to a branch, you have to switch to it.

Switching to another branch

You already know the command to switch between versions. Well, we will use the same command to navigate between branches. Simply use “git checkout” with the name of the branch as a parameter.

Therefore switch to the data-preprocessing branch created earlier. Remember you can always use the git branch command to get a list of all current branches, if you forget the branch's name.

To switch to the branch, run the command;

git checkout data-preprocessing

You should get something similar to this

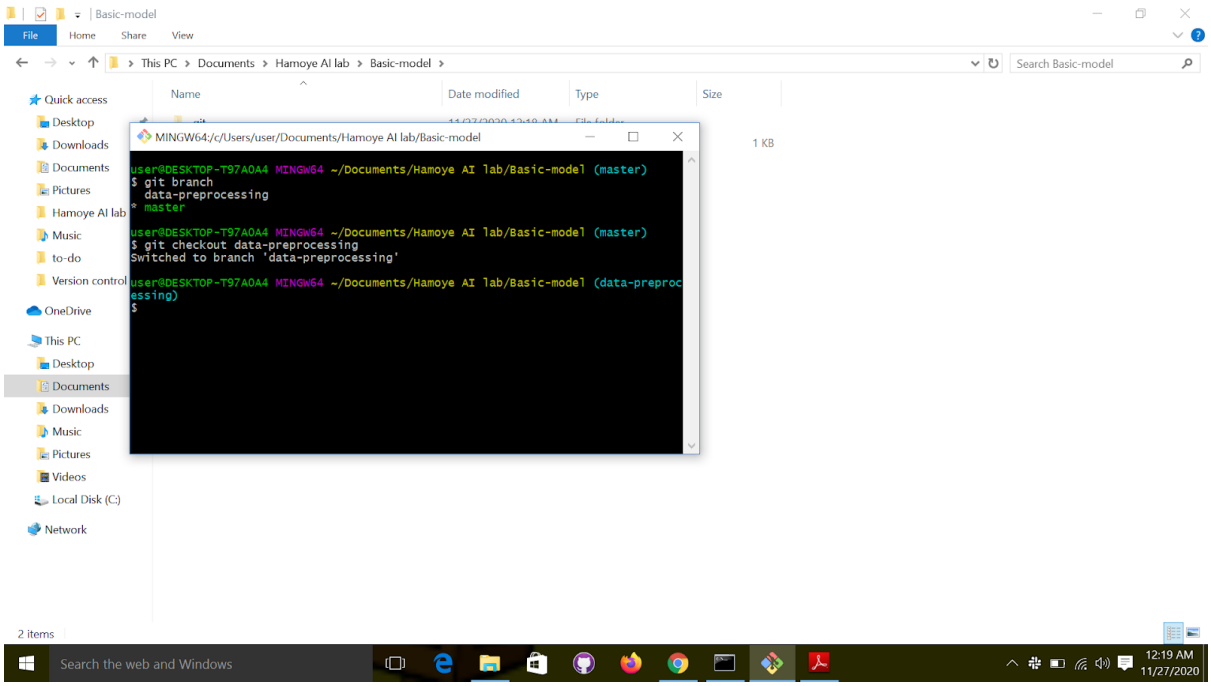

Figure 34: switching between branches.
Now that you have created and switched to the data preprocessing branch, let's try something fun:

1. Using the basic model repo
2. Open "git bash"
3. Create and switch to the data-preprocessing branch.
4. On this branch lets create a processing.py file using any text editor.
5. Save and exit
6. Stage and commit changes
7. "Push" "branch" to the remote repository. To do this, run the command;

git push origin data-preprocessing

Syntax is git push origin branchnam

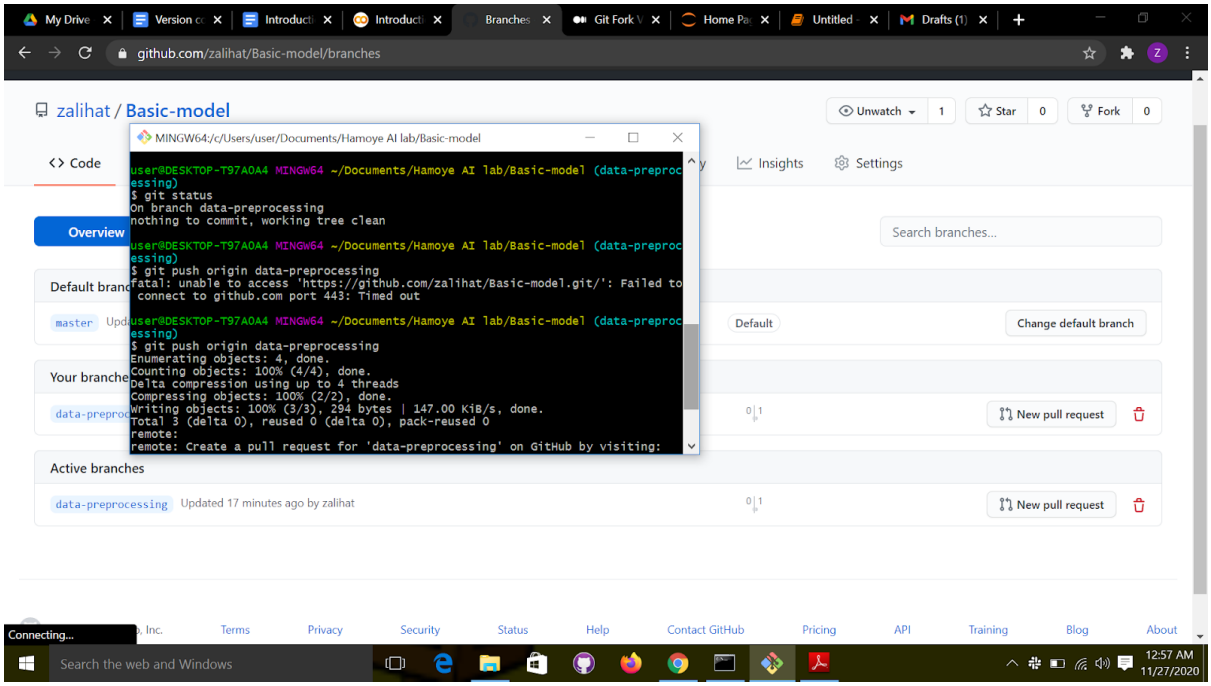

Figure 35: pushing a branch to a remote repository

If you check the remote repository, you will see that there are now two branches available; the master and data-preprocessing branch.

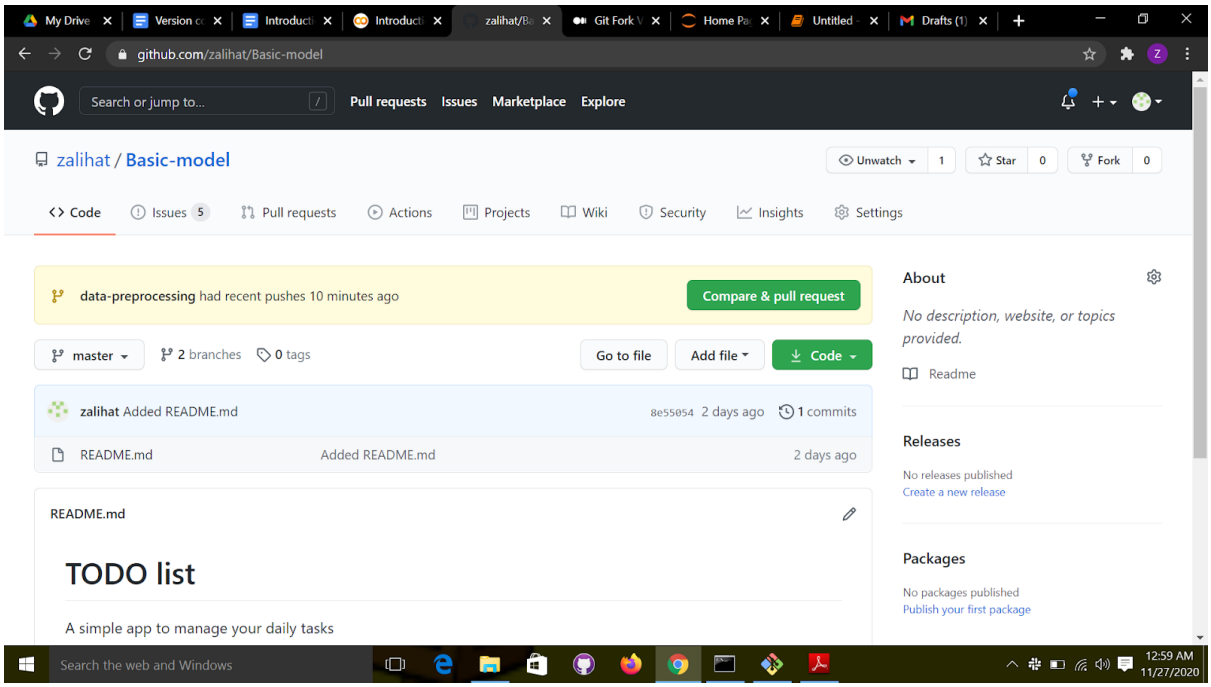

Figure 36: Remote repository's branches

Here is the cool part, if you switch back to the master branch, the preprocessing.py file will not be there. The reason is that the changes were made to the branch created, not to the main project. How then do we bring the two together? The answer is pull request, and merge.