Collaborating on GitHub

Must know tools for Data Scientists

Ujjwal Dalmia Sep 11, 2020 · 9 min read ★

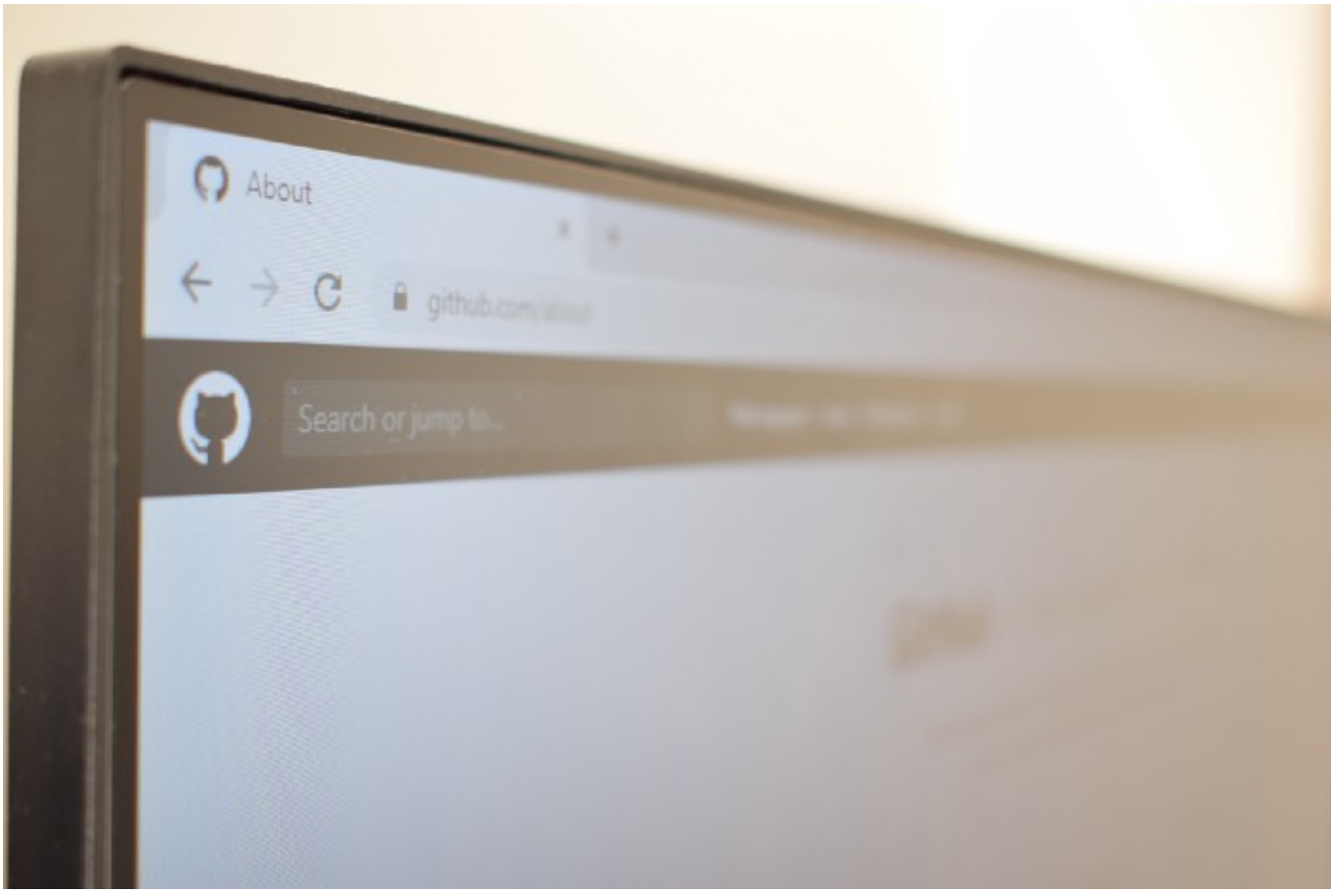


Photo by Richy Great on Unsplash

In the last tutorial, we learned about the basics of GitHub. In this tutorial, all those basics will come together and we will get to experience the real power of GitHub which

any of these, please read through the last tutorial first.

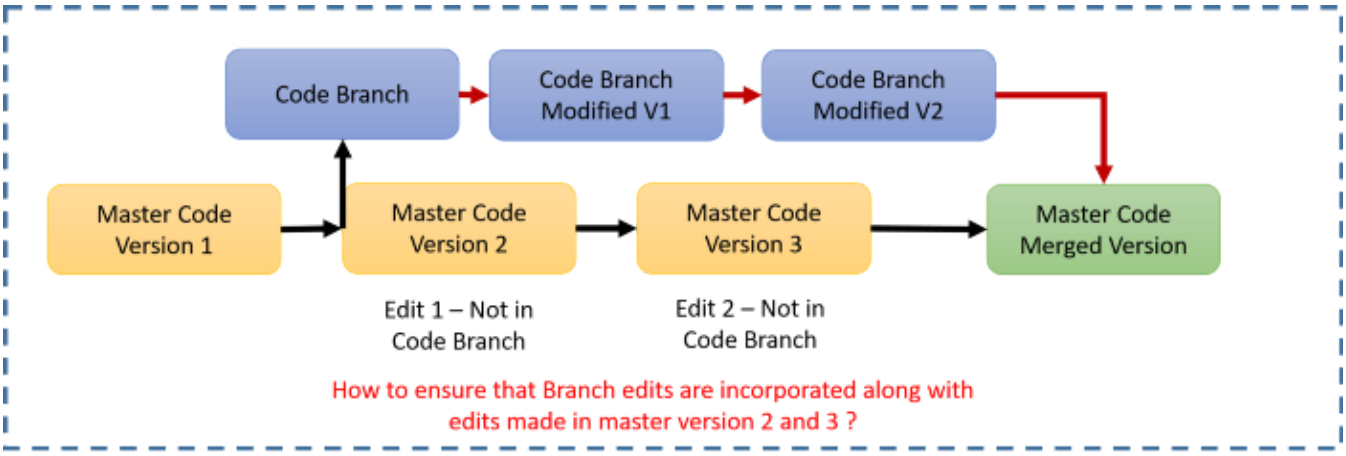
Defining Collaboration

To appreciate the solution, it is important to appreciate the problem. Read through the following to understand the scenario where GitHub can come in handy.

You are a lead data scientist who is working on a machine learning project with a team of junior data scientists. Since you are the project owner you are responsible for maintaining the "***master***" **version** of the project code.

In your team, every data scientist is working individually to improve the model prediction. Therefore, they have created their local versions or **alternate branches of the "*master*" version**.

Let's assume that **you have made some edits** to the "***master***" version of your code and **a member of your team comes up with his own edited version** and claims that he has considerably improved the model prediction. A visual of this scenario is as follows:



Sample Scenario (Image by Author)

You now want to update the "***master***" version of the code but a number of conflicts can arise. Mentioning a few below:

- The edits you have made to the "***master***" version are probably not there in your team member's branched version
- The branched version might have some variable treatments which are different when compared with the "***master***" code

GitHub comes in handy

Solution Time

I am sure that the above scenario must have made you appreciate the problem at hand. The question is, how will GitHub help?

First Things First

Before we jump onto the solutions, let's understand some basic terminologies we will be using throughout this tutorial:

- **Collaborator** — A developer who is **formally added to the project repository** and is given the **push access** (access to modify the content of the repository) is called as a contributor or collaborator
- **Fork —** Forking in GitHub is like **copying someone else's repository** into your account. Generally, when you want to use an **open-source project** created **by some other developer** and **you are not a contributor**, you **fork** it and get access to their repository.
- **Branch** — Generally, developers **use different branches for maintaining different modules of the project**. Another common scenario that warrants the use of branches is when **multiple members of the team want to work on the same piece of code.** This is when each one can have its own branch. By default, **each newly created repository has a central branch** named "***master***" branch.
- **Pull Request** — A pull request is created **to merge a branch with the "*master*" branch**. The request goes directly to the project owner and he/she can work with the branch contributor to accept/ reject the edits.

Scenarios

To address the problem defined above, we will run through 2 scenarios which you can encounter when actively collaborating on GitHub:

- In the first scenario, we are assuming that you want to contribute to a repository (project) where **you are not added** as a contributor. In this scenario, we will assume that **there are no edits made to the "*master*" branch after you have forked the project** (in the sample scenario diagram shown above, yellow boxes of Version 2 and 3 **don't exist**).

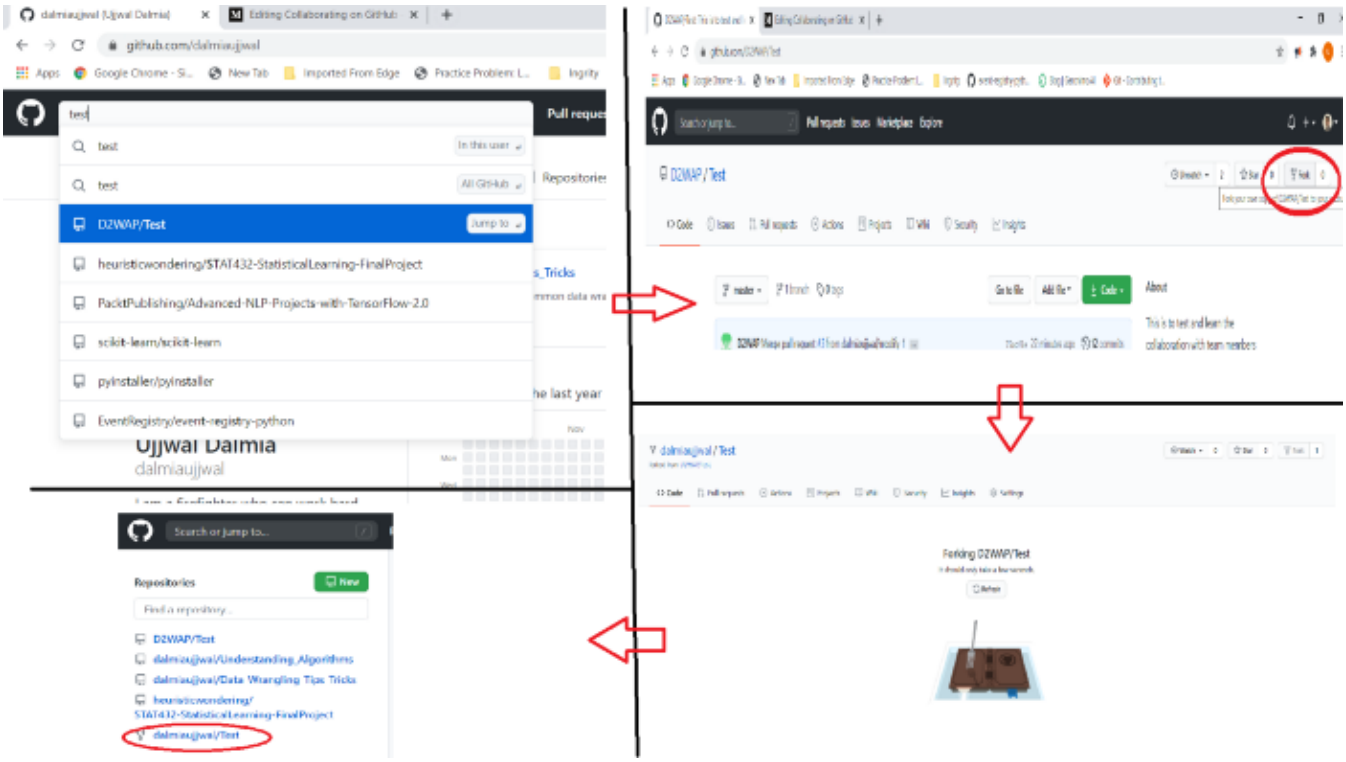
there are edits made to the "*master*" branch after you have created your own branch (in the sample scenario diagram shown above, yellow boxes of Version 2 and 3 **exist**).

Let's look at the workflow of the above scenarios.

Scenario 1 — Not a contributor

When you are **not a contributor** to a repository, you are **not allowed to push edits** from your local system to GitHub. The workflow steps to collaborate in a scenario like this are as follows:

- **Forking the repository —** Since we are not a collaborator to the repository, we will **first fork the repository**. To fork the repository, login to your account and search for the repository you are interested in. Go to the repository GitHub page and click on the "***Fork button***" as shown below:



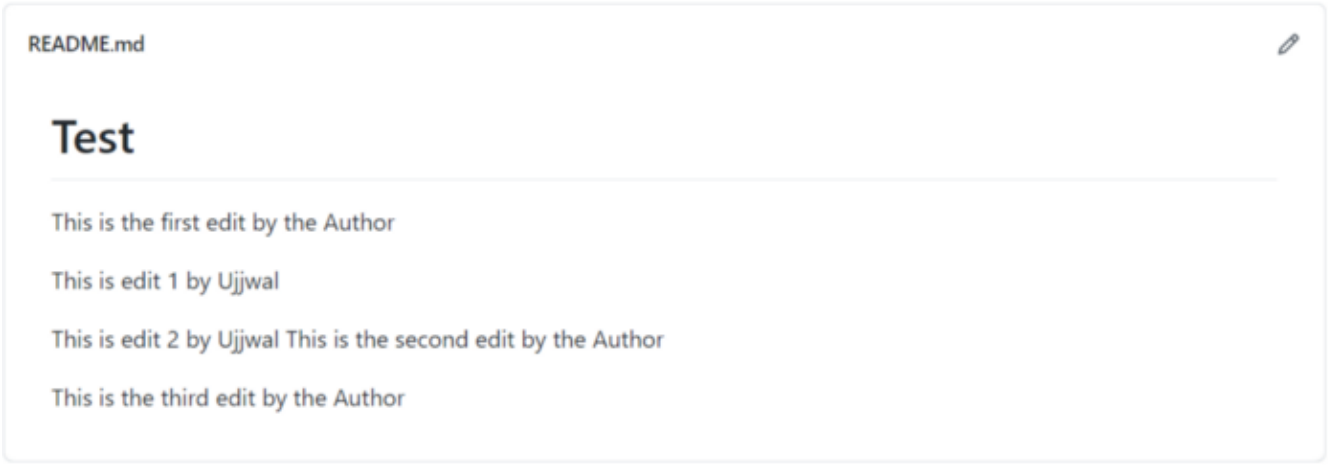
Forking (Image by Author)

In the above 4 screenshots, we are **searching for the repository** (D2WAP/Test). We went on to the **repository page and pressed the fork button** on the top right. The forking process happened and the **repository got forked** in our account.

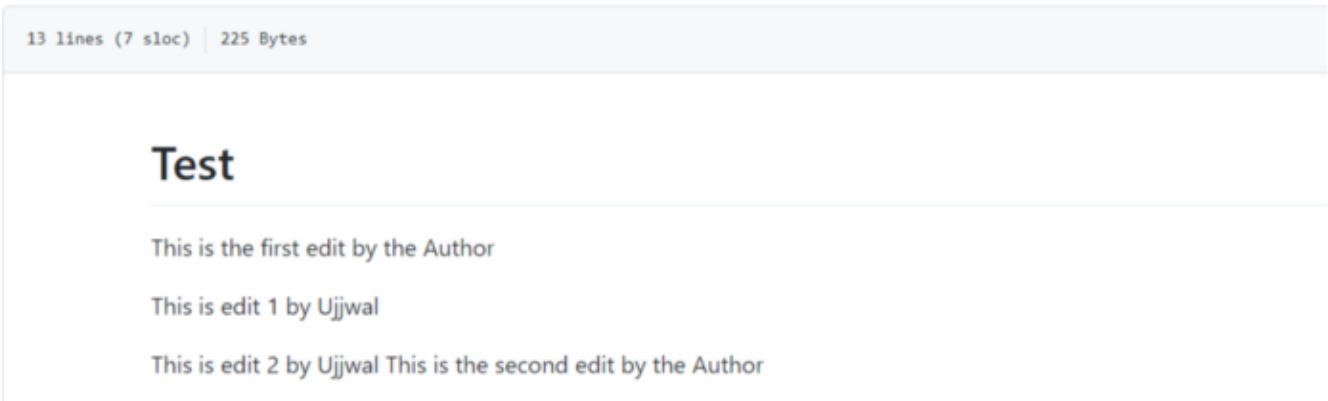
the last tutorial. Since **the forked repository is part of your account, you will have push access and you can push your edits to the forked repository**. Please note, in this case, the **cloning should be done after forking** because, If done before forking, your locally cloned repository would point to the **other developer's account** where you will not **have the push access**.

- **Modify the code**, **commit and push** — The next step is to **edit the codes in the cloned repository**, **commit the changes,** and **push them to your forked repository**. Given you have a fork, you **can push the edits directly to the "master" branch** or **can create a new branch** also. It is recommended that we **do the edits in a new branch only** but to keep this scenario simple, we are pushing the edits directly to the "***master***" branch**.** We will look at the new branch creation in scenario 2. To demonstrate the process, I have edited the ***read me file*** of the forked repository. The difference between the forked and original repository can be seen from the screenshot below:

Original Read Me File

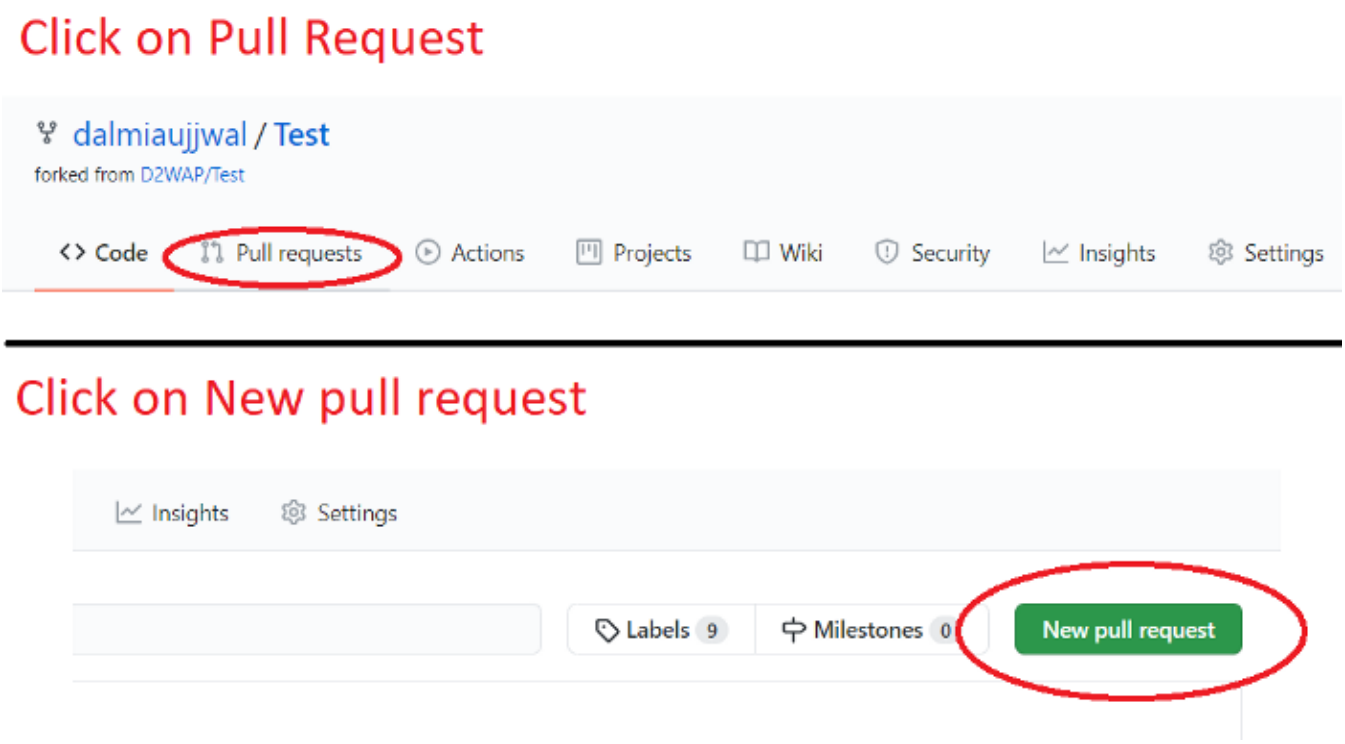


Updated Read Me File in the Forked Repository



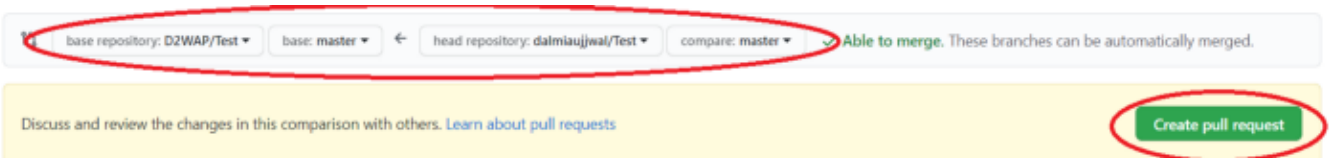
Editing Read Me File (Image by Author)

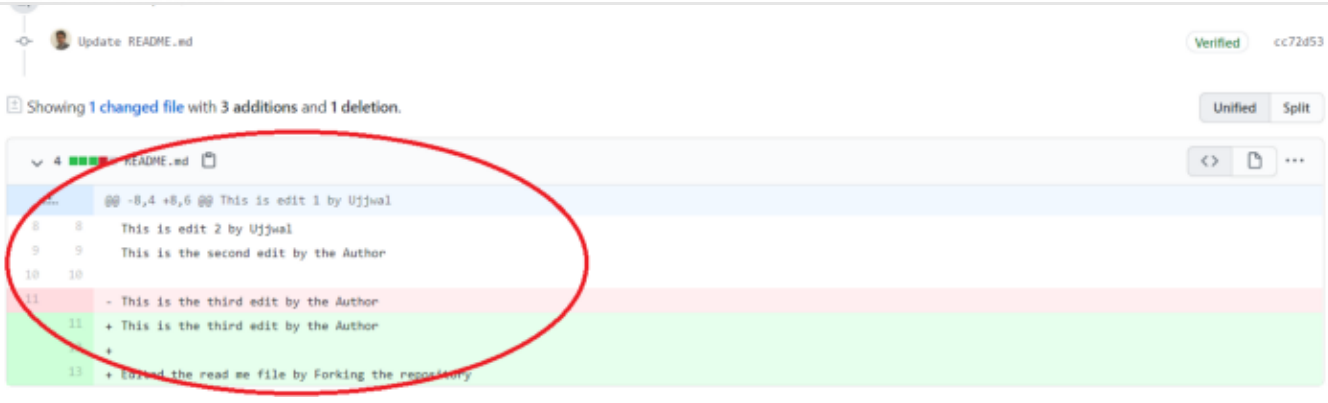
- **Create New Pull Request —** Now that you have made necessary edits to the codes (***read me file* in our case**), it is time to create a pull request. **Click on "*Pull request*" option** and go to the pull request screen. On the top right of the screen click on "***New pull request*".**



Creating New Pull Request (Image by Author)

- **Select base and head repository** — Once you click on "***New pull request***", you will reach a page where you can **select the base and head repository** along with **corresponding branches**. Think of base as the destination repository and head as your forked repository. Since we have made the edits directly to the "***master***" branch of the forked repository, **for both head and base branches select "*master*"**. Once the selections are made, GitHub will show you the **comparison between the 2 versions** you are trying to merge. Again, click on "***Create pull request***".





Selecting Source & Destination Repository (Image by Author)

- **Add Comments** — Once you have clicked on "***Create pull request***", you will reach the **comments page** where you can add **appropriate comments** for the source repository developer. Once again **click on "*Create pull request*"** and your pull request will be sent to the source developer. **Notice the message from GitHub saying there are no conflicts**. This validates our assumption that there were no edits made in the source after we have forked the repository.

Comments Section (Image by Author)

- **Merge by Author** — Once you have submitted the pull request, the author receives it, and given there are no conflicts, he/she can approve it straight away. To merge, the author will go to the pull request tab and click on "***Merge pull request***". That's it, you have successfully contributed to the source repository

Merge Approval Process (Image by Author)

Scenario 2 — You are a contributor

Since in this scenario we are assuming that you are added as a contributor by the repository owner, you will have the access rights to push the edits directly on to the source repository. With this assumption, let's go through the workflow:

- **Clone** — Since you can now directly push the edits on to the main repository, there **is no need for a fork**. Go ahead and clone the repository. The process for this is explained in the last tutorial

- **The source file is edited** — Refer to the screenshots below to see the edits made to both the source file and the branched file. For demonstration purposes, we are modifying the ***read me file***.

Editing Read Me File (Image by Author)

- **Drag & Drop**— Assuming that you have made necessary edits, you can go to the original repository page on GitHub and drag & drop the edited file on the page. Screenshot added for your reference:

Drag & Drop (Image by Author)

- **Create a new Branch** — Once you have dropped the edited file on the repository page, you will get options to either directly edit the "***master***" branch or to create a new branch. In this scenario, we will go with the recommended approach of creating a new branch. When the corresponding radio button will be selected, GitHub will ask you to name the branch. Once named, click on "***Propose changes***". Every time you want to backup or version control your work, you can do this step. Just make sure that if you are continuing to version control the edits before generating a pull request (explained in the next step), you are doing it on the same branch. The screenshots are added below for your reference:

Creating New Branch (Image by Author)

- **Create New Pull Request** — Once you will propose the changes, you can create a pull request just the way it was explained in scenario 1. Note that, since we have made the edit directly to the source repository, you are not required to select the base and head repositories but only the branches to be merged. Also, since the 2 branches have been edited in parallel while merging a conflict is raised by GitHub. Screenshots added for your reference:

Pull Request With Conflict (Image by Author)

- **Resolving Conflicts** — This is where the repository owner and the branch collaborator can get into a discussion and together resolve the conflict. On GitHub, this can be done by clicking on "***Resolve conflict***" and can be initiated by both the repository owner or the branch owner. In the screenshot below, notice the edits which are leading to conflicts. Which edit to keep and which to reject can be

Resolving Conflicts (Image by Author)

- **Merge Pull Request** — Once the conflicts are resolved, you will reach the Pull request screen with the Conflicts message is replaced with the Merge Pull Request. Click on "***Merge pull request"*** and then click on "***Confirm merge***" along with any message if you want to document. That's it, your edits are merged with the "***master***" branch. Screenshots added for your reference:

Merging Pull Request (Image by Author)

- **Rechecking the "*master*" branch** — Once merged, you will see the "***master***" branch is updated with the edits from the branch.

Closing note

There are a lot of other functionalities associated with git & GitHub but those are for some other day.

Equipped with this knowledge, collaborate with your project teams, and leave the worry of version control to GitHub. Hope this tutorial was helpful and you learned something new.

Will try and bring more interesting topics in future tutorials.

HAPPY LEARNING !!!

Github Programming Data Science Machine Learning Analytics