+-

**Lab: Getting started with GitHub**



**Estimated time**: 30 minutes

In this lab, you will get started with GitHub by creating a GitHub account and project. You will then add a file to the project using the GitHub web interface.

**Objectives**

After completing this lab, you will be able to:

1. Describe GitHub
2. Create a GitHub account
3. Add a project/repository
4. Create and edit a file
5. Upload and commit a file

**GitHub overview**

Before we learn to use GitHub, let’s first review Git. *Git* is an open-source version-control system for software development. It is simply a place where you can collect all your folders and files for a project. This collection of folders and files is usually called a repository.

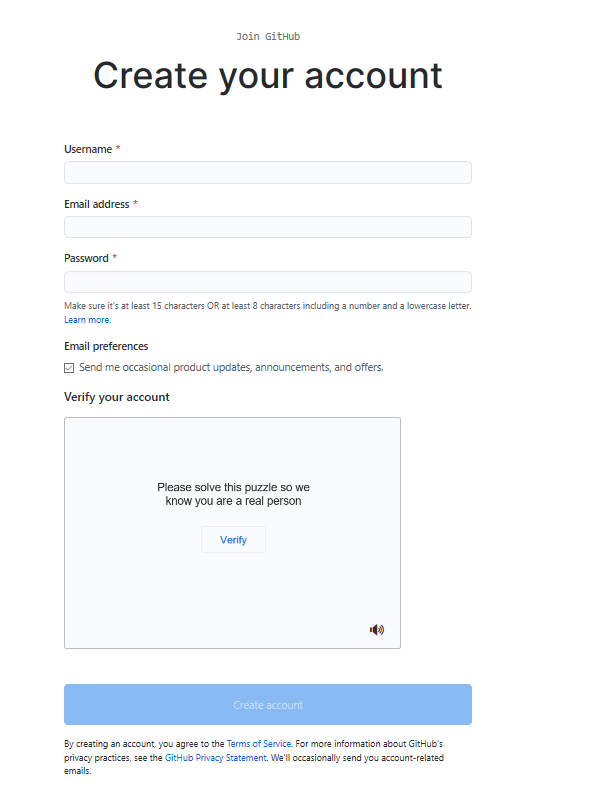
*GitHub* is a Git repository hosting service, but it adds many of its features. While Git is a command-line tool and you also need to host and maintain a server using the command line, GitHub provides the Git server for you, as well as a web-based graphical interface. It also provides access control and several collaboration features, such as wikis and basic task management tools, for every project.

GitHub provides cloud storage for source code, supports all popular programming languages, and streamlines the iteration process. GitHub includes a free plan for individual developers and for hosting open-source projects.

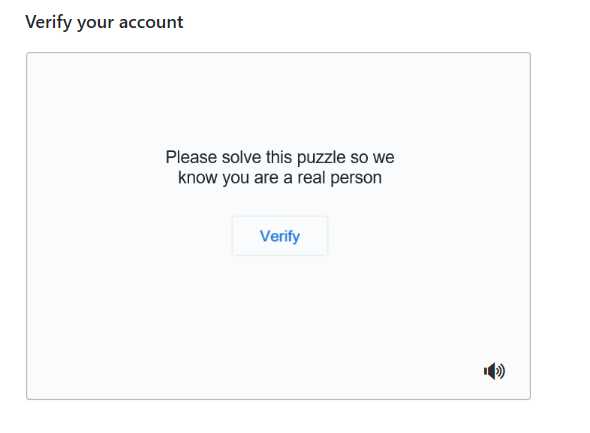
**1. Creating a GitHub account**

To create an account in GitHub, complete the following steps:

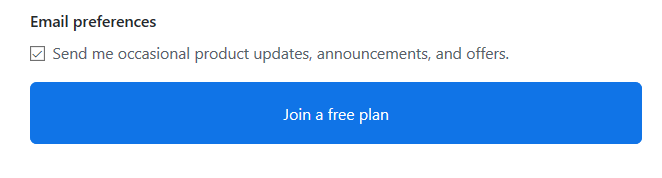
1. Go to the [Join GitHub page](https://github.com/join) and create an account. **Note**: If you already have a GitHub account, log in now.
2. Provide the necessary details to create an account as shown below. When you have finished, click **Create account**.



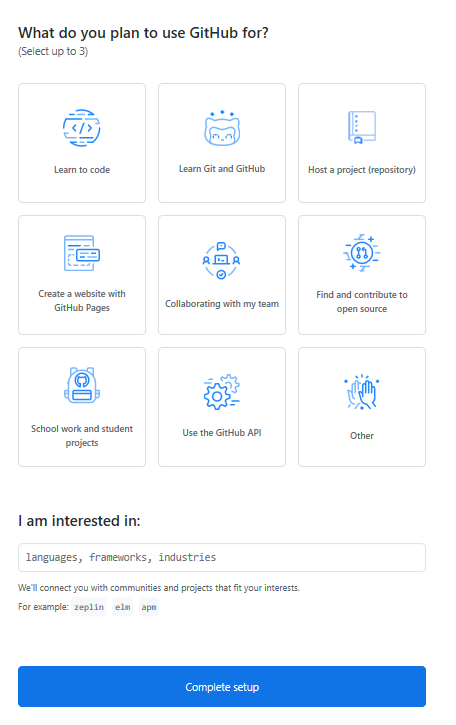
1. Click **Verify** to verify the account and then click **Done**.



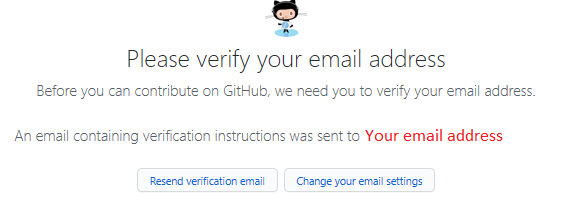
1. After you have verified, click **Join a Free Plan**.



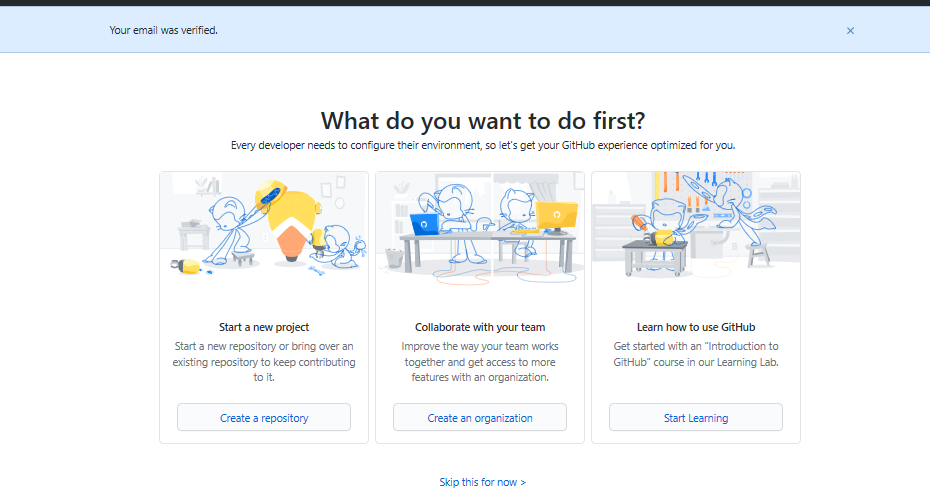
1. Select your reasons for joining GitHub and click **Complete Setup**.



1. You will receive a verification email from GitHub. Click the enclosed link to verify your email. **Note**: If you do not receive a verification email, click **Resend verification email**.



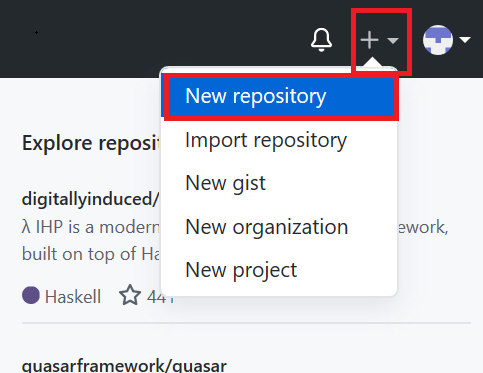
When you have verified your email, you will see a screen that looks like this:



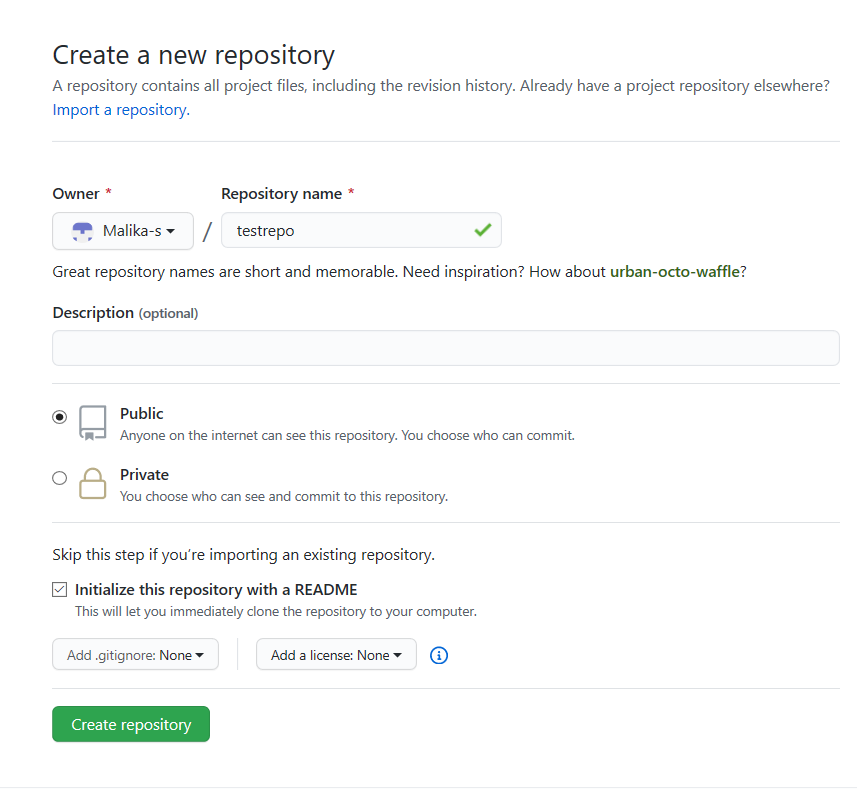
**2: Add a project / repository**

To add a new repository, complete the following steps:

1. At the top right of the GitHub home page, click on the “+” icon and select **New repository**.



1. Enter a repository name and select the **Initialize this repository with a README** check box.



1. Click **Create repository**. The repository is created and its home page is displayed.

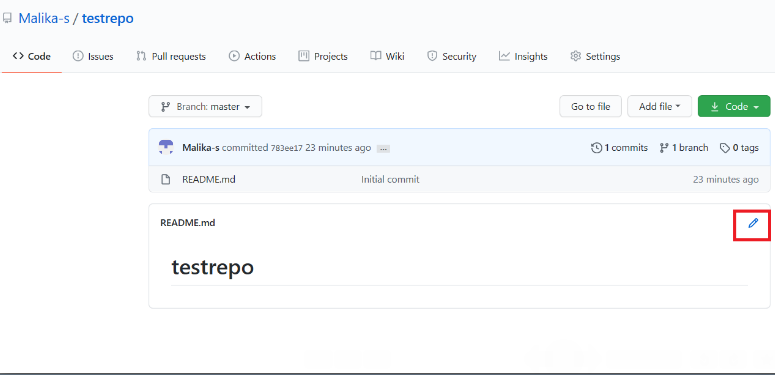
Next, you’ll start editing the repository.

**3: Create and edit a file**

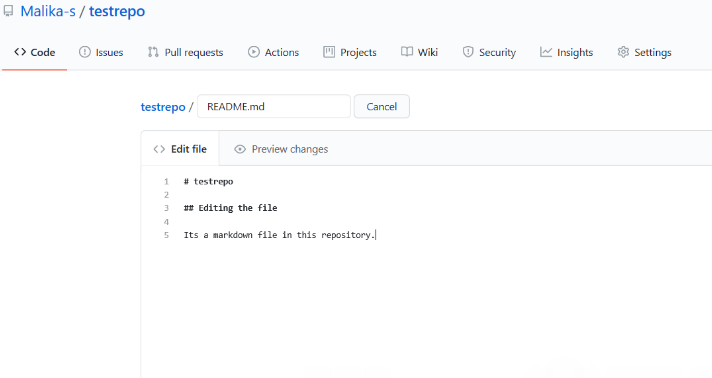
**Edit a file**

Although you will normally create a file before you edit it, in this case, the README.md file has already been created for you. To edit that file, complete the following steps:

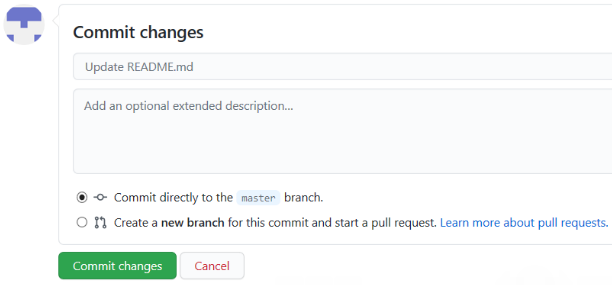
1. Your repository root folder contains just one file: README.md. Click the pencil icon at the right to edit the file.



1. Add some text to the file.



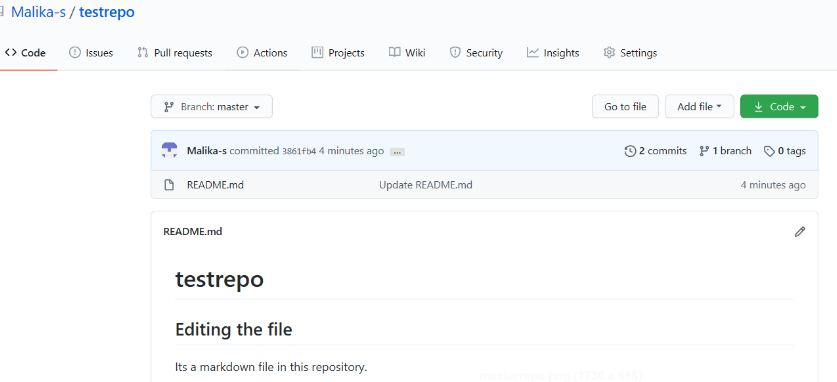
1. Scroll to the bottom of the file and click **Commit changes**.



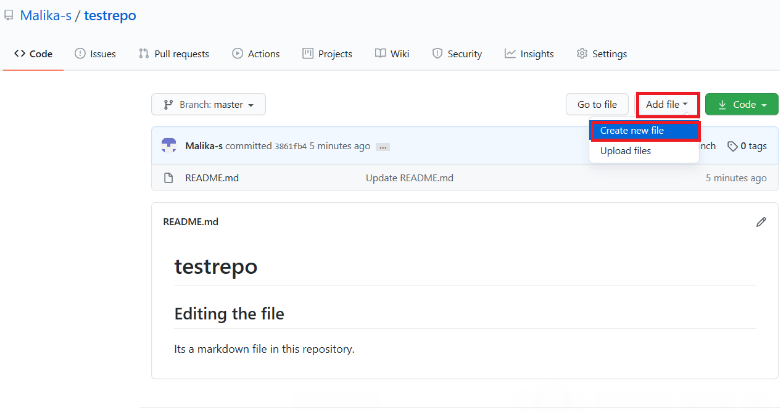
1. Confirm that the text you added to the file has been saved.

**Create a new file**

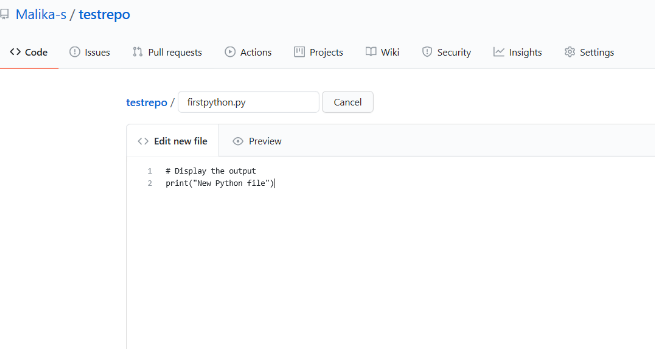
1. Click on the repository name to go back to the main branch, similar to this repository called testrepo.



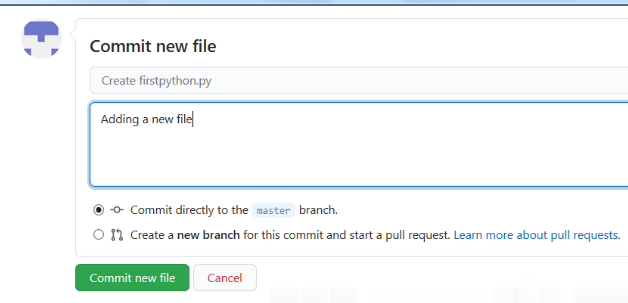
1. Click **Add file** and select **Create new file** to create a new file in the repository.



1. Enter a file name and extension; for example, firstpython.py. Add the lines of code displayed in the following image to your file:



1. Scroll to the bottom of the page. You can optionally add a description of your update (for example, “Adding a new file”). Click **Commit** to create your new file.

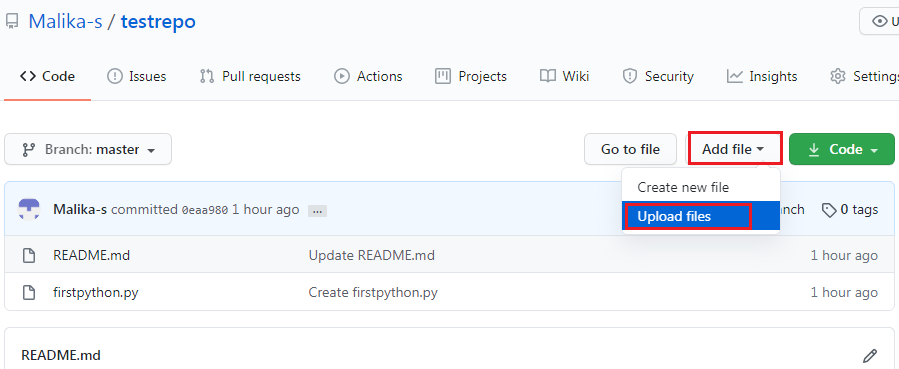


Your file is now added to your repository and the repository listing shows when the file was added or last revised.

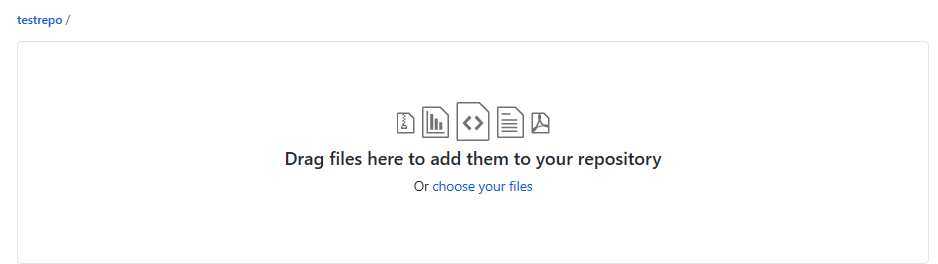
**4. Upload and commit a file**

To upload a local file and commit it to your repository, complete the following steps:

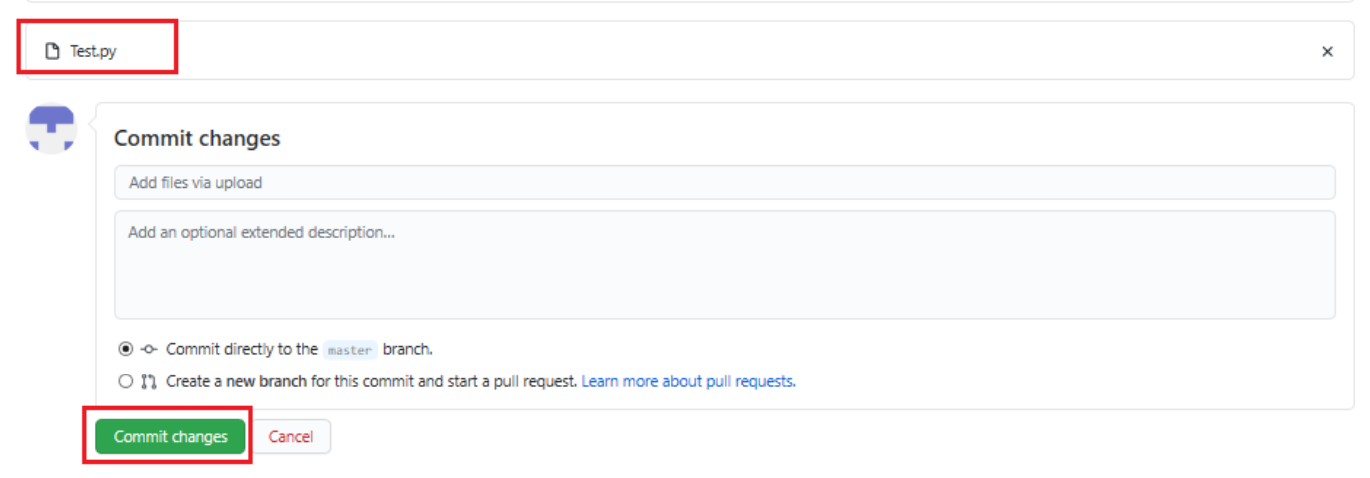
1. On your repository page, click **Add file** and then select **Upload files** to upload a file.



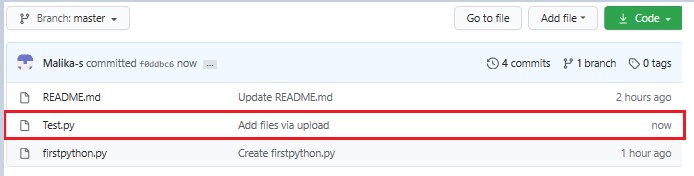
1. Click **choose your files** and choose a file from your computer. You can upload any file (for example, a .txt, .ipynb, or .png file) to the repository.



1. When the file finishes uploading, click **Commit changes**.



Your file is uploaded to the repository.



**Summary**

Congratulations! In this lab, you have learned how to create a new repository, add a new file, edit a file, upload a file, and commit your changes. We encourage you to continue to update your repository to become familiar with the processes that you have learned.

**Tutorial details**

**Authors:**

* Romeo Kienzler
* Malika Singla

**Other contributors:**

* Rav Ahuja
* Upkar Lidder

**Changelog:**

| **Date** | **Version** | **Changed by** | **Change Description** |
| --- | --- | --- | --- |
| 2020-08-18 | 0.6 | Upkar Lidder | Updated GitHub Overview section |
| 2020-07-16 | 0.5 | Malika Singla | Spell check, step number added |
| 2020-07-14 | 0.4 | Rav Ahuja | Changed logo, updated effort, title, intro, objectives, added Authors and Changelog |
| 2020-07-13 | 0.3 | Malika Singla | Added to GitLab and made some formatting changes, added objectives, etc. |
| 2020-07-03 | 0.2 | Malika Singla | Ported to markdown and added GitHub account signup, new screenshots, etc. |
| 2020-06-30 | 0.1 | Romeo Kienzler | Drafted initial version |

**© IBM Corporation 2020. All rights reserved.**

+-



**Estimated time**: 15 minutes

## Objectives

After completing this lab you will be able to:

1. Create a branch
2. Commit changes to a child branch
3. Open a pull request
4. Merge a pull request into the main branch

## Prerequisites

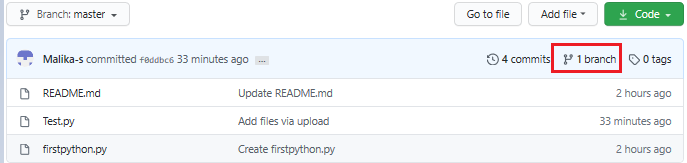
This hands-on lab requires you to have created a GitHub account and added a project to it, as covered in the [Getting started with GitHub](https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-CD0101EN-SkillsNetwork/labs/GitHubLabs/GitHub1_Getting_Started.md.html) lab.

*NOTE: In the past the default branch in your GitHub repo used the name master. Effective Oct 1. 2020, all new GitHub repositories use the more inclusive term main as the name of the default branch instead of master.*

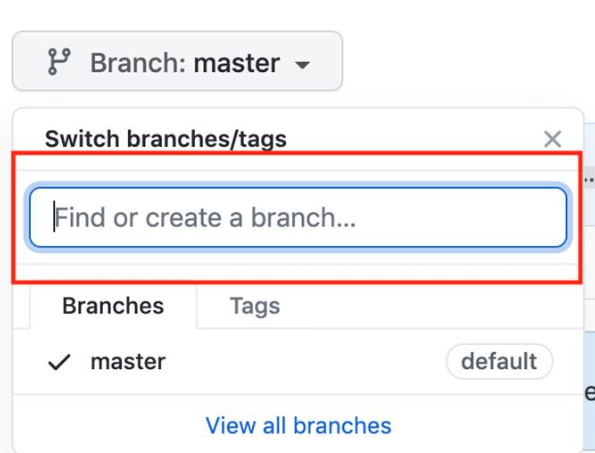
## 1. Create a branch

You can create or delete branches using your repository’s GitHub web page. To add a branch to your repository, complete the following steps:

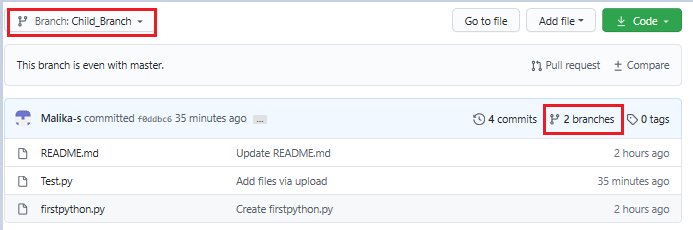
1. Go to you repository’s main page. Note that when you created your repository, the **main** branch was created for you:



1. At the top of the file list, locate the **Branch** drop-down menu. (By default, the menu displays **Branch: main**.) Click the drop-down menu, type the name of the branch you want to create, and press Enter on your keyboard.



Your repository now has two branches: **Main** and **Child\_Branch**. You can click the drop-down menu to see your branches.

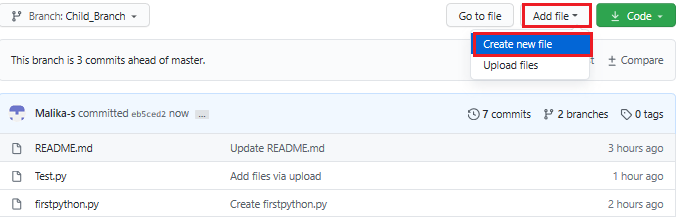


Any files that were in the **main** branch have now been copied to **Child\_Branch**. Note that when you add or edit a file in **Child\_Branch**, that change will not automatically be made in the **main** branch.

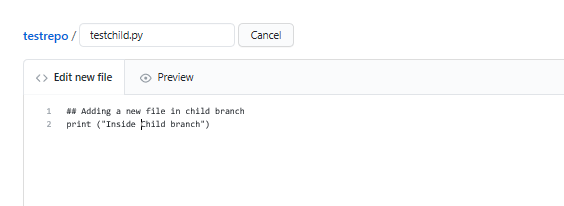
## 2. Add a file to a branch

To add a file to your new branch, ensure that **Child\_Branch** (or whatever name you gave your branch) is displayed in the **Branch** drop-down menu and complete the following steps:

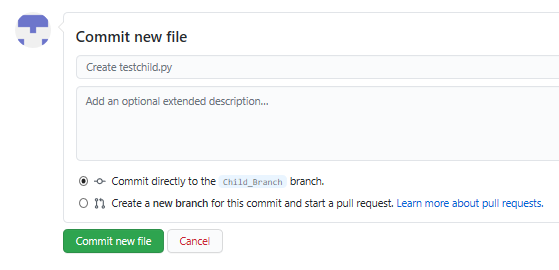
1. Click **Add file > Create new file** to create a file in the repository.



1. Type a name and extension for the file – for example, testchild.py – and add the following lines to the body of the new file:



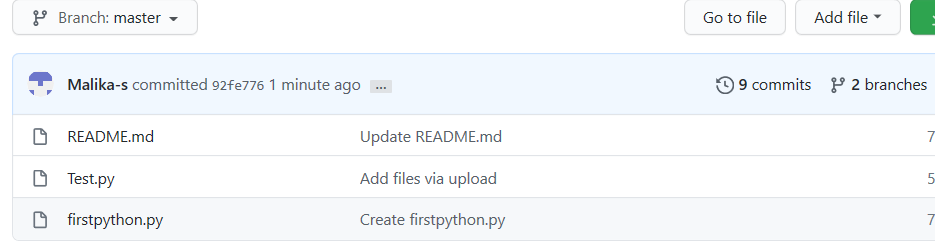
1. Scroll to the bottom of the page, add a description of the file you are about to add (note that the description is optional), and click **Commit**.



The file is added to your child branch.

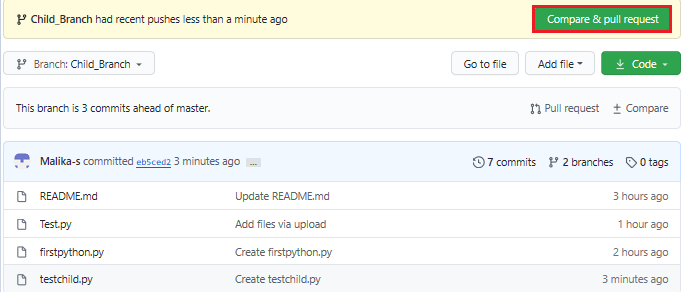
## 3. Open a pull request

The file that you added to your child branch is not automatically added to the **main** branch. (You can check this by using the **Branch** drop-down menu to go to the **main** branch; note that there is no testchild.py file in the file list):

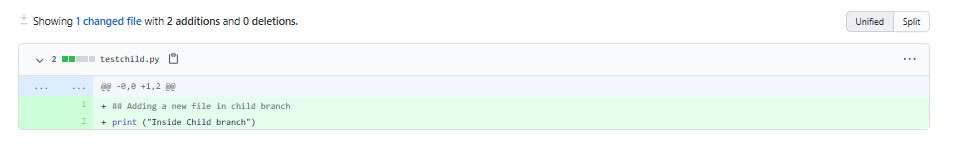


You can also compare the two branches and open a pull request, which will enable you to copy the changes that you’ve made in the child branch – in this case, adding a new file – to the **main** branch.

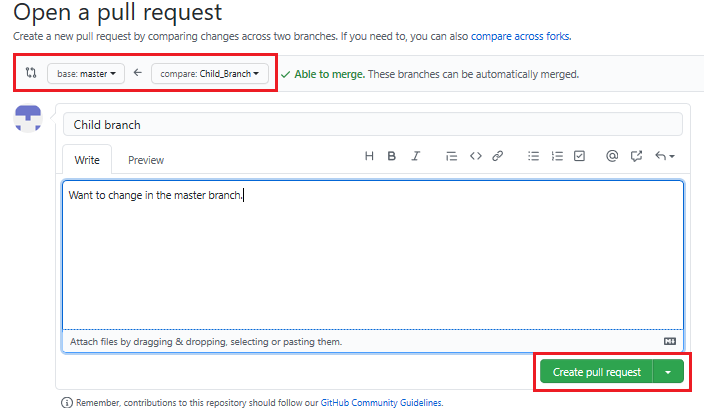
1. In **Child\_Branch**, click the **Compare & pull request** button.



1. Scroll to the bottom of the page and note that there is **1 changed file** listed.



1. Scroll up and note that GitHub is comparing the **main** and **Child\_Branch** branches and that there are no conflicts between the two. Optionally, you can add a comment to the pull request. Click **Create pull request**.

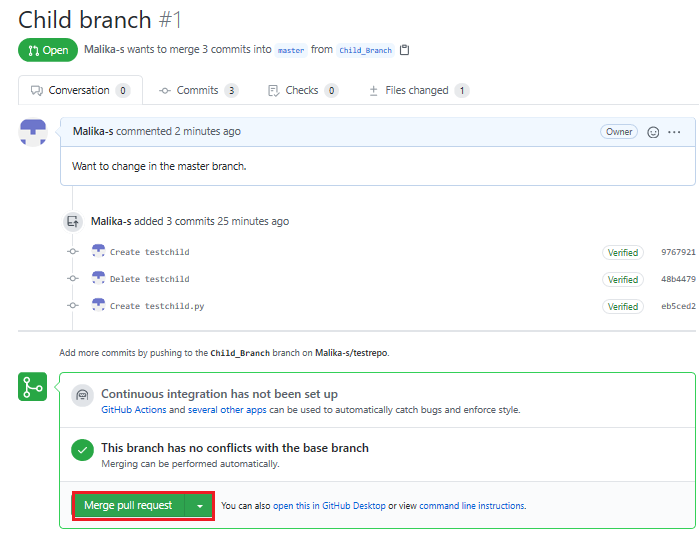


The pull request is created and can now be merged by a repository administrator.

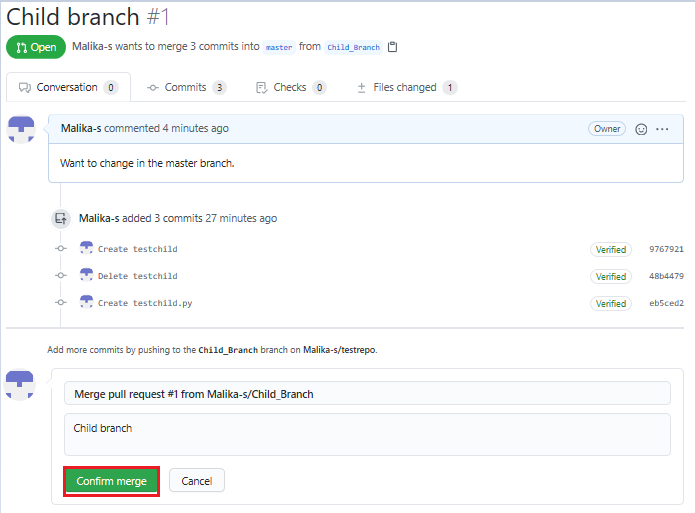
## 4. Merge a pull request

To merge a pull request into a project, complete the following steps:

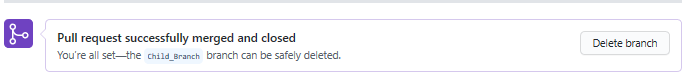
1. Click the **Pull requests** tab. A list of pending pull requests is displayed.
2. Click the pull request that you want to merge into the main project. If you are satisfied with the changes, click **Merge pull request** to accept the pull request and merge the updates. (You can add a comment if you choose.)



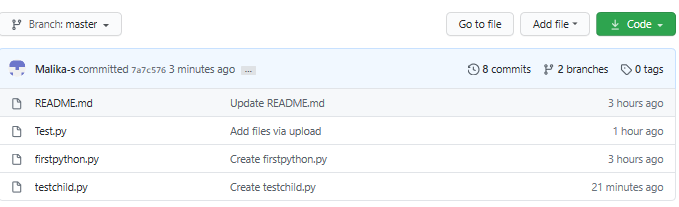
1. When you click **Merge pull request**, a **Confirm merge** button is displayed. Click that button to complete the merge.



The pull request has now been merged successfully. Note that you can delete the child branch because your changes have been incorporated into the **main** branch.



Check the list of files in the **main** branch to confirm that it now includes the file that you added in the pull request.



## Summary

Congratulations! You’ve now learned how to create a branch, edit and commit changes in that branch, open a pull request, and merge the pull request into your main project. We encourage you to continue to experiment with branches and pull requests to become more familiar with the concepts and processes.

## Tutorial details

**Author:**

* Malika Singla

**Other contributor:**

* Rav Ahuja

**Changelog:**

| **Date** | **Version** | **Changed by** | **Change Description** |
| --- | --- | --- | --- |
| 2023-01-30 | 0.5 | Lavanya Rajalingam | Fixed the link and minor corrections |
| 2022-01-14 | 0.4 | Rav Ahuja | Added note about main branch |
| 2020-07-16 | 0.4 | Malika Singla | Spell check, steps added |
| 2020-07-14 | 0.3 | Rav Ahuja | Changed logo, updated title, intro, objectives, added Effort, Authors and Changelog |
| 2020-07-13 | 0.2 | Malika Singla | Added to GitLab and made some formatting changes, added objectives, etc. |
| 2020-06-30 | 0.1 | Malika Singla | Drafted initial version |

Forking and Cloning GitHub repos

# Forking and Cloning GitHub repos



If you are wondering what is the difference between fork and clone, don’t worry - that is a natural question to have when you are starting out with GitHub. In this reading we will answer this question as well as clarify when should you use which and how do you perform the fork and clone operations. As well as demystify repository (repo) terminology like local, remote, origin and upstream. We will also walk through a typical workflow involving forking, cloning, making changes and keeping various repos in sync using fetch, push, pull and pull requests.

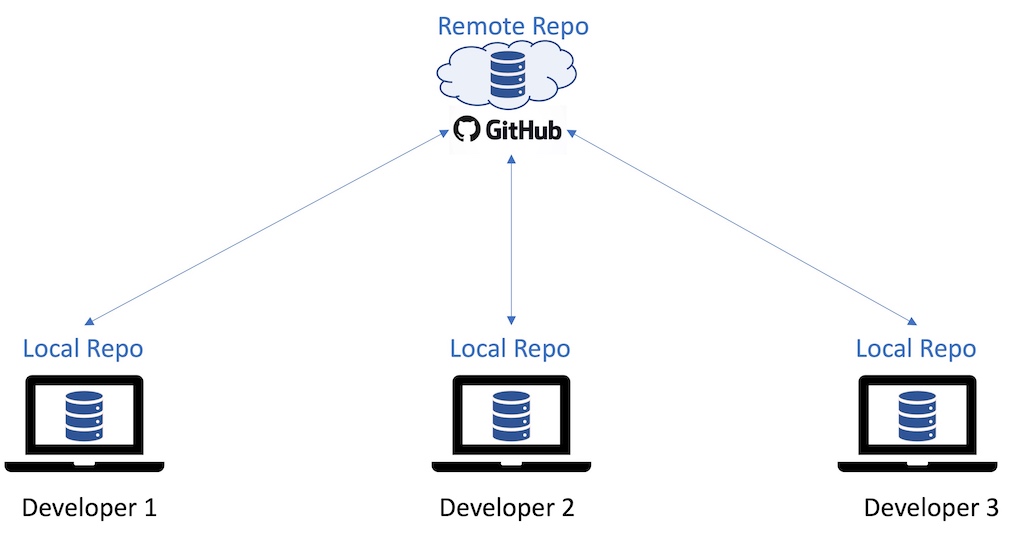
## Objectives

After completing this reading, you will be able to:

* Differentiate between forking and cloning repositories
* Explain how repos can be forked and cloned
* Determine when you should fork vs. clone
* Describe terminology such as local, remote, origin and upstream respositories
* Explain how to keep distributed repos in sync using push, fetch and pull

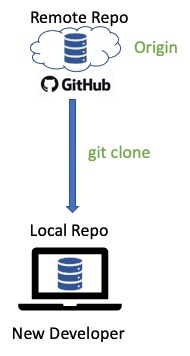
## Distributed Repositories

Distributed source version control systems like GitHub allow for projects to have multiples developers collaborating on its codebase in parallel. A project may reside on GitHub.com as a public or private repository. Each developer working on the project may have their own copies of the repository on their computers. A copy of the repository on a developer’s computer is local to that developer, and hence that developer also refers to that repo as their local repo. The copy of the repository on GitHub.com is on a remote server, and hence for each developer that is a remote repo.

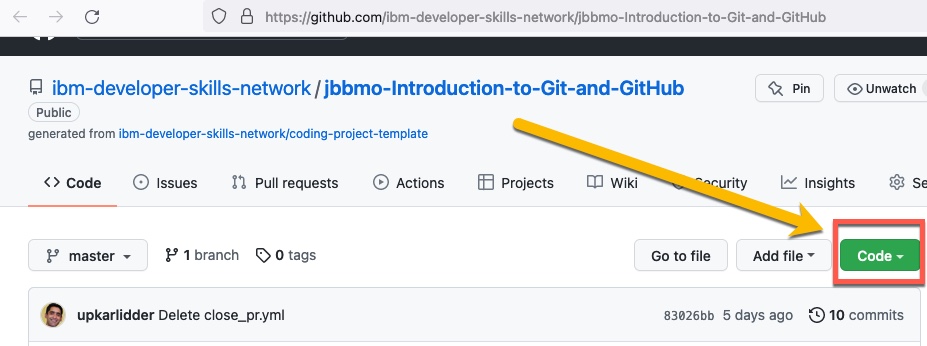


## Clone

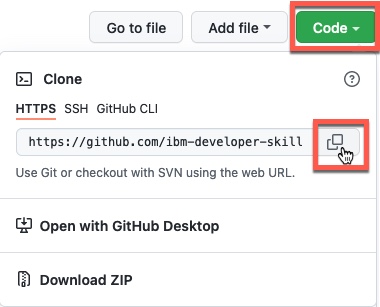
Let’s say a new developer now joins the team to collaborate on the project. This developer can create a identical copy of the remote repo using the git clone operation. The remote repo from which the project is originally cloned from is also referred to as the origin.



Any repo in GitHub can be cloned by going to the repo and clicking on the Code button.



You will then have an option to get the entire codebase of the remote repo in several ways, including the ability to copy the HTTPS URL, and then specifying the copied URL to perform git clone URL command from your local machine.



## Creating Branches and Synchronizing Changes

After cloning the repo to the local machine, a developer can start making changes to the codebase. This could be for tasks like adding features and enhancements or fixing bugs. Typically the developer would use git branch to create a branch, e.g. feature1-branch, make that branch active using git checkout and make changes within that branch - such as by adding or editing files. The developer saves their changes often within the branch by using git add followed by git commit.

Once the changes for a particular branch are complete, rather than merging to the main branch directly, it is often a good practice to push the new branch with changes to origin where other developers/reviewers can test/review the changes before merging the branch to main.

*NOTE: Since in this scenario the feature1-branch was developed by a new developer on the project, that developer may not have the access to merge their branch with main in origin. In fact, in many projects, only the project maintainers or admins are allowed to merge to the main branch, or in some sames a peer review may be required. In order to request that your changes be reviewed and merged with the main branch, many projects require that a Pull Request (PR) be submitted. Whereas, in some cases, e.g. if you are a lone developer on the project, this PR step may be omitted and you could merge and push your changes directly if you have write access to the origin repo.*

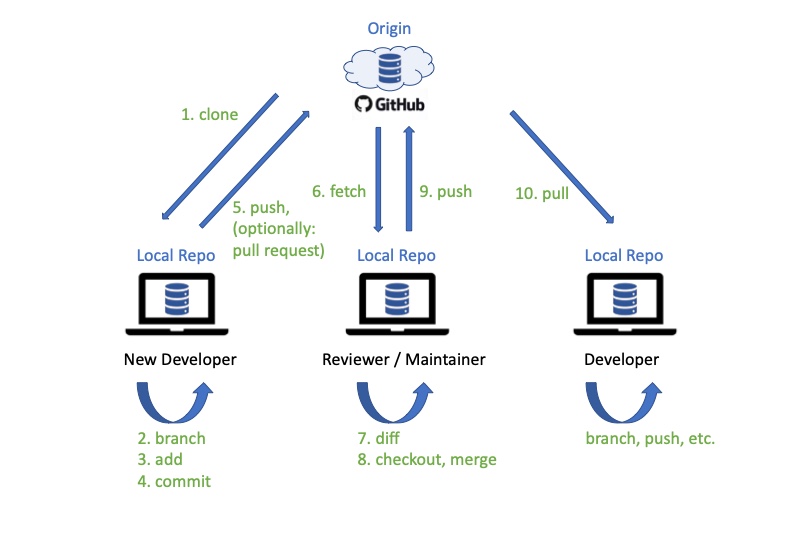
Every once in a while, a developer may want to get the latest copy of the repo from origin to serve as the base for making changes or reviewing changes by others. For example, this may be the case after the changes in feature1-branch have been pushed to origin and the peer developer wants to review the code. The git fetch command can be used for this purpose.

The git diff command can help others reviewing your code to to identify and compare the changes. Once a peer reviewer or project maintainer has reviewed the changes, and is satisfied, the reviewer will git checkout the main branch and then git merge the new feature1-branch, which can then be deleted. After the branch is merged locally, the reviewer can git push the updated main branch back to origin.

*NOTE: The git-remote -v command can be used to check which remote repos you are synchronzing push and fetch changes with.*

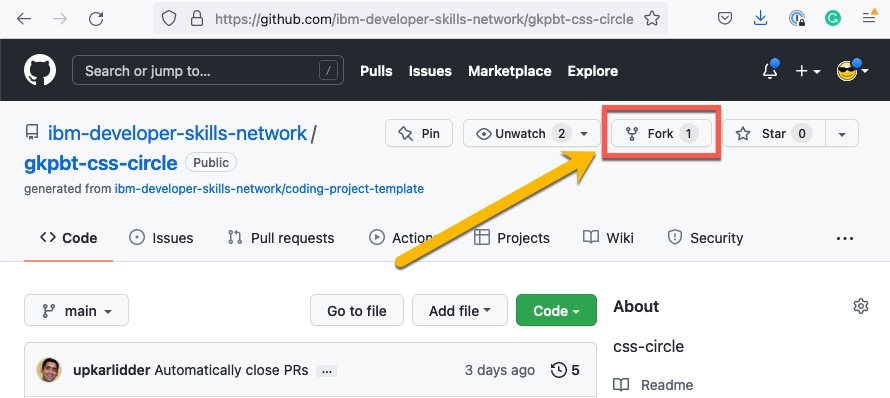
Another option for getting the latest copy of the repo is to use the git pull command. The pull command in effect is a combination of fetch and merge. That is, using this single command, you can both fetch and merge the changes into your local repo. For example, another developer who wants to use the updated codebase with the feature1 changes that have been merged to main branch in origin, can use the git pull command to fetch the updated codebase from origin and merge with his/her local codebase before starting development on a new feature.

This clone->branch->merge workflow described here can be summarized in the following diagram.



## Fork

If a developer wants to create a derivative project with another project as the starting point, or work on a project using a separate or independent clone, the developer can chose to fork a project. You can fork any public project by going to its GitHub project page and clicking on the Fork button towards the top of the page.

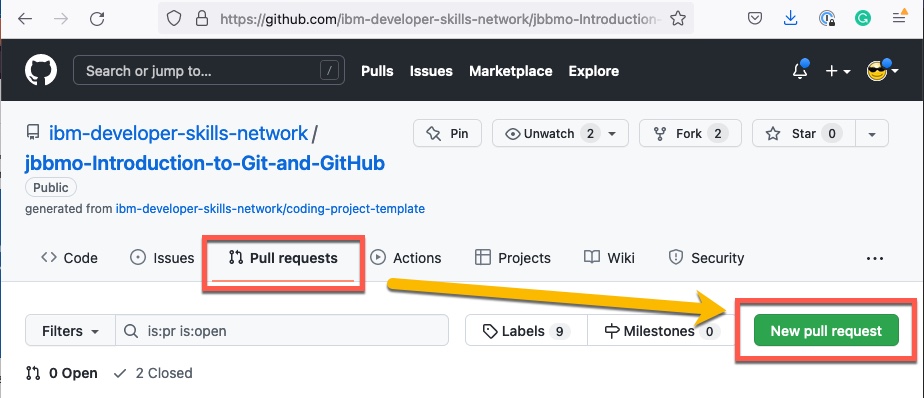


*NOTE: The fork option is available only using the web interface and there is no git command to create a fork. You can however use a git clone workaround - indicated at the bottom of this reading if interested.*

The project from which you create the fork is called the upstream project.

Once a project has been forked, the developers with access to the fork can work on updating and making changes to the fork using the same workflow as described previosuly i.e. the forked copy of the project now becomes the origin and developers with access to origin can create clones of it on their local machines, where they can create and merge branches, and synchronize changes with the origin using pull and push.

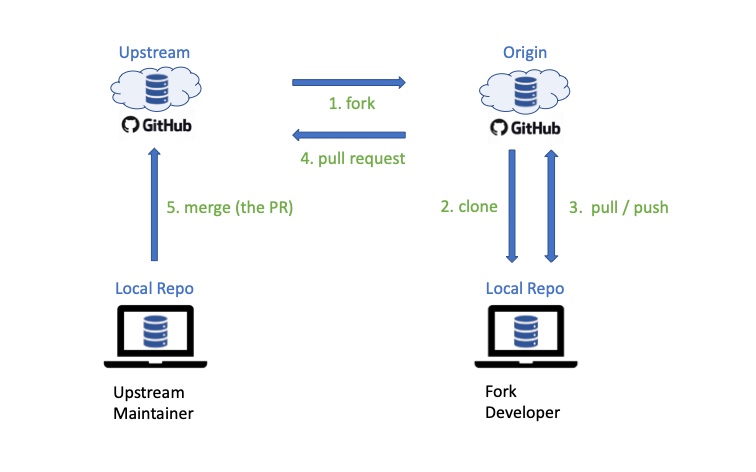
However it is important to note that the synchronization of changes using merge and push can only be done with repos that the developers have write access to i.e. in this case their fork of the project i.e. the origin from which they create their local clones. But what if a developer wants contribute their changes back to the upstream project that they do not have write access to? In this case they can submit a pull request or PR with their proposed changes. A pull request can be opened by going to the project’s homepage, navigating to the Pull Requests tab, and then clicking on New Pull Request.



*Note: the term Pull Request should not be confused with the git pull command that you use to fetch and merge the latest codebase into your local repo. A Pull Request, as the name implies, is merely a request to review and pull your proposed changes. As part of the PR, you provide details of the proposed changes and your implementation.*

The maintainers of the upstream project can review the changes in the PR and decide to merge them or not. In some cases they may provide feedback (by commenting in the PR) or ask the submitter of the PR to perform some conflict resolution such as by applying their changes to the latest codebase and resubmitting the PR.

This fork->clone->pr workflow described here is summarized in the figure below.



## When to fork or clone?

By now you should be familiar with the difference between fork and clone. So let’s summarize when you should clone vs. fork. Typically if you have access to a project repo e.g. as part of a team developing a codebase collaboratively, you can clone the repo and synchronize changes from your local copy of the repo using pull and push.

If however there is a public project that you want to contribute to but do not have write access to, or use a public project as a starting point for your own project, you can fork the project. Then work with the forked codebase by cloning it to your machine and collaborating with your development team working on the fork using the pull-push synchronization with your fork of the project. But if you want to contribute your changes back to the upstream project (the original project that you forked from), you can submit your changes using a pull request.

## Summary

In this reading you learned that:

* A clone is essentially your copy of a project that you can make changes to.
* You can create local copy of a project by using the git clone command.
* The project from which you clone from is called the origin.
* You can pull updates from the origin and push your changes back to it.
* A fork is an separate copy of a project that you can make changes to independently of the original project
* The project that you fork from is referred to as the upstream project
* You can suggest changes back to the upstream project by submitting a Pull Request (PR)

*FYI: Although the usual workflow to start with the codebase of another project is to first fork it and then clone the fork, you may be tempted to simply clone the upstream project since it is quite convenient to do so from your local machine using the git clone command. If you do so, you will note that the project you clone from will by default become the origin repo. But since you likely don’t have write access to the upstream repo that you cloned from, you will not be able to push your changes to it. Don’t worry. You can easily rename the origin to upstream using the command git remote rename origin upstream and then add a new origin using git remote add origin <url> to point to the URL of a new GitHub repo that you have created or have access to, and use that repo for making your changes to the fork’s code.*

## Author(s)

#### Rav Ahuja

### Other Contributor(s)

#### Richard Ye

## Changelog

| **Date** | **Version** | **Changed by** | **Change Description** |
| --- | --- | --- | --- |
| 2022-01-19 | 1.0 | Rav Ahuja | Initial version created |
| 2022-01-27 | 1.1 | Richard Ye | Fixed Typos |
| 2023-04-03 | 1.2 | Lavanya Rajalingam | Updated new SN Logo |