**National University of Computer & Emerging Sciences, Karachi**

**Computer Science Department**

**Fall 2024, Lab Manual - 10**

|  |  |
| --- | --- |
| **Course Code: SL3001** | **Course: Software Development and construction** |
| **Instructor:** | **Yasir Arfat** |

**Lab # 10**

# Aims and Objectives

# People use GitHub to build some of the most advanced technologies in the world. Whether you’re visualizing data or building a new game, there’s a whole community and set of tools on GitHub that can help you do it even better. Who is this for: new developers, new GitHub users, and students. What you'll learn: We'll introduce repositories, branches, commits, and pull requests. What you'll build: We'll make a short Markdown file you can use as your profile README.

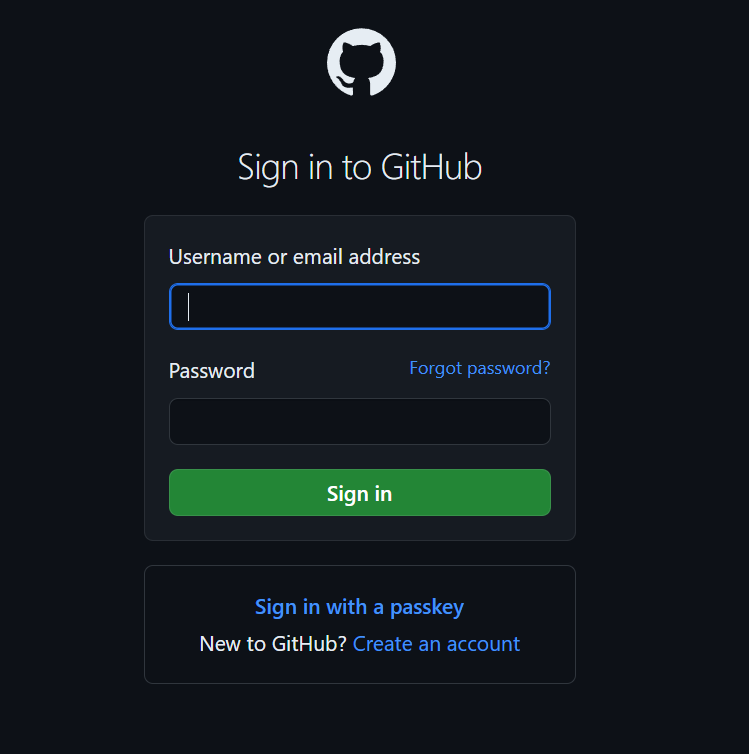
# Introduction

GitHub is a web-based platform that provides version control using Git, allowing multiple people to collaborate on projects, track changes, and manage code repositories efficiently. It’s widely used by developers for both open-source and private projects.

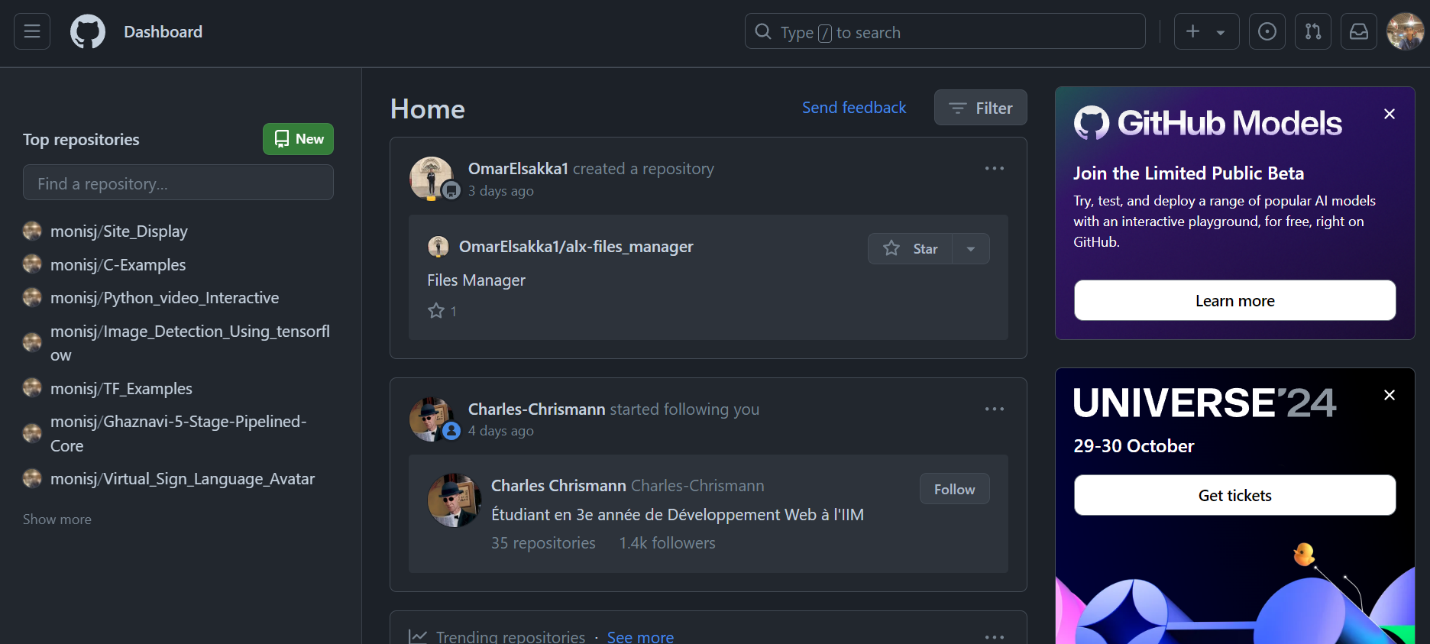
# In this Lab, you will: Create an account on GitHub followed by a repository. Create a branch. Commit a file. Change the readme file Open a pull request. Merge your pull request. Understand and implement markdown language for readme files

# Create an account on GitHub

To get started first login to GitHub (if you already have an account).



If not, then click on the **create an account** option and enter the necessary credentials to get started. Once logged in you should see a screen something like.



Of course, mine has many repositories so when performing your screen would look a bit different. On the top left section where you see the new button, click on it and you should see a window something like this.

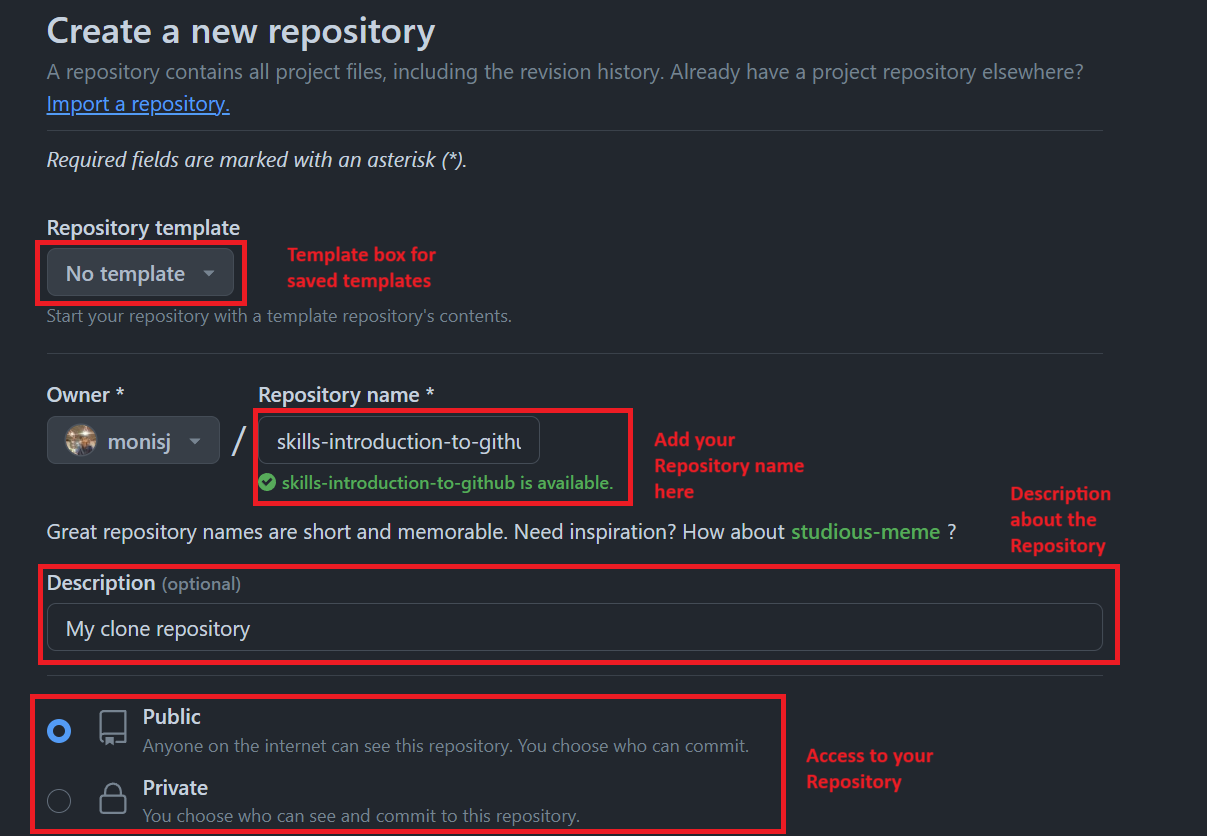
A screenshot of a computer

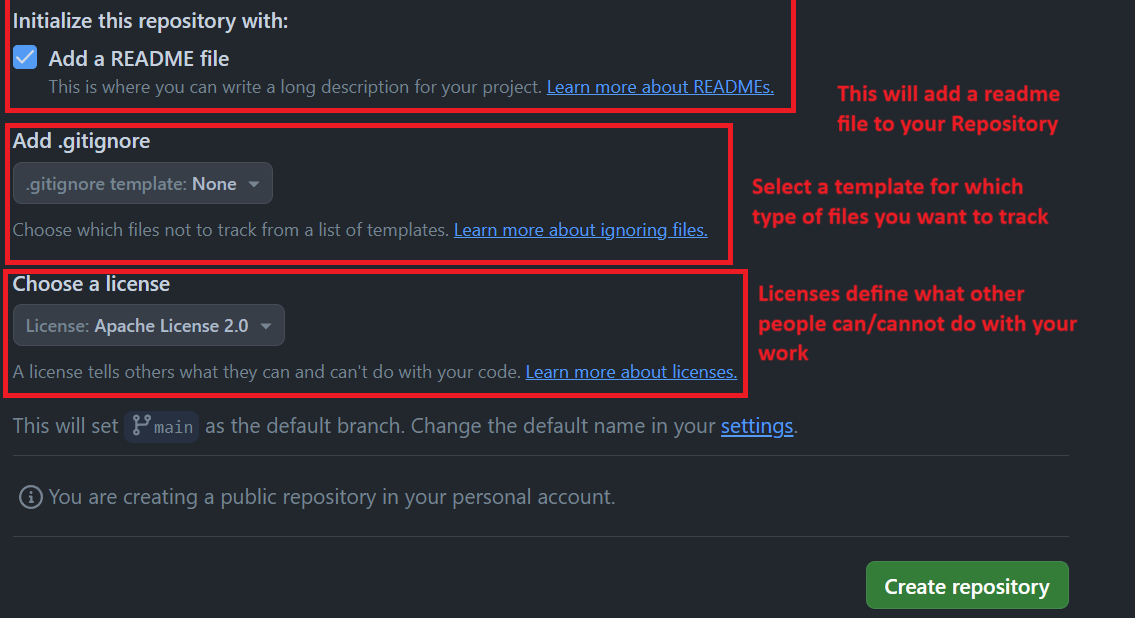
Description automatically generated

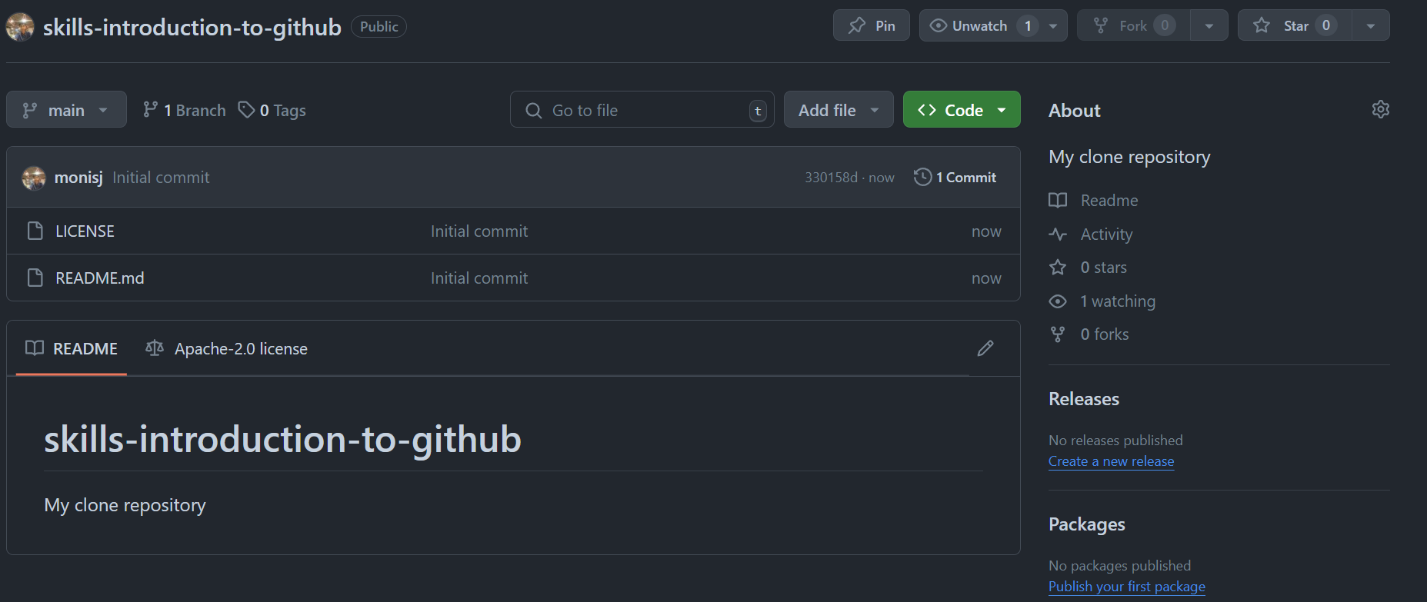
Which leaves us to another heading on how to create a repository.

# Create a repository

**What is a repository?**: A [*repository*](https://docs.github.com/get-started/quickstart/github-glossary#repository) is a project containing files and folders. A repository tracks versions of files and folders. For more information, see "[About repositories](https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories)" from GitHub Docs.

  
When creating a repository you are required to fill/select at least three sections of the page. The first red box is if you want to use already saved templates (These are already made repositories with some files and branches already stored in them). The second box is your repositories name, You can name your repository anything you like but stick to proper naming conventions and name your repository according to what project you are working on, GitHub will indicate you if your repository name is unique or not. The Third box is optional and is used to display a small description about your repository. Lastly the access box and this can be crucial depending on your project. If you intend to display your project to everyone then you would need to select public access but if you choose to hide your project before releasing it to the public a private option is available and only you and those who are contributing to your project can see the repository.

  
Lastly there also exists three more optional options when creating a repository. The First check box is a detailed description about your project, sort of like an actual MUST-READ FILE when installing pirated software. We will cover this later down in the manual on how we can modify this. The second box is for tracking which types of files to ignore, choose none and all files will be tracked otherwise you can select which type of files to exclude from tracking. Lastly a license is not compulsory for this lab but mandatory when releasing your project to the world as it defines the boundaries of what the end user can/cannot do with your work. There are many versions of this license so read and explore accordingly.  
Once all the necessary boxes are filled click on the **Create repository** button.

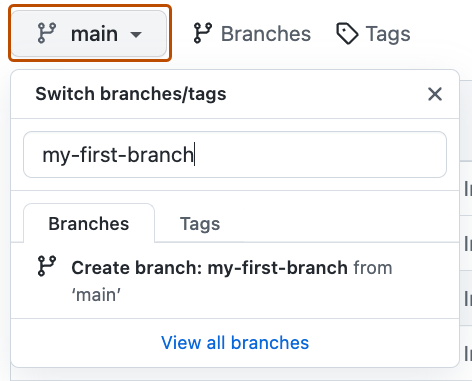


You should see a screen something like this. The bottom section is the readme file, the Top right is the description for the repository and the center is where you will see your files which you have uploaded. Right now, there are only two files as we created a license and a readme file.

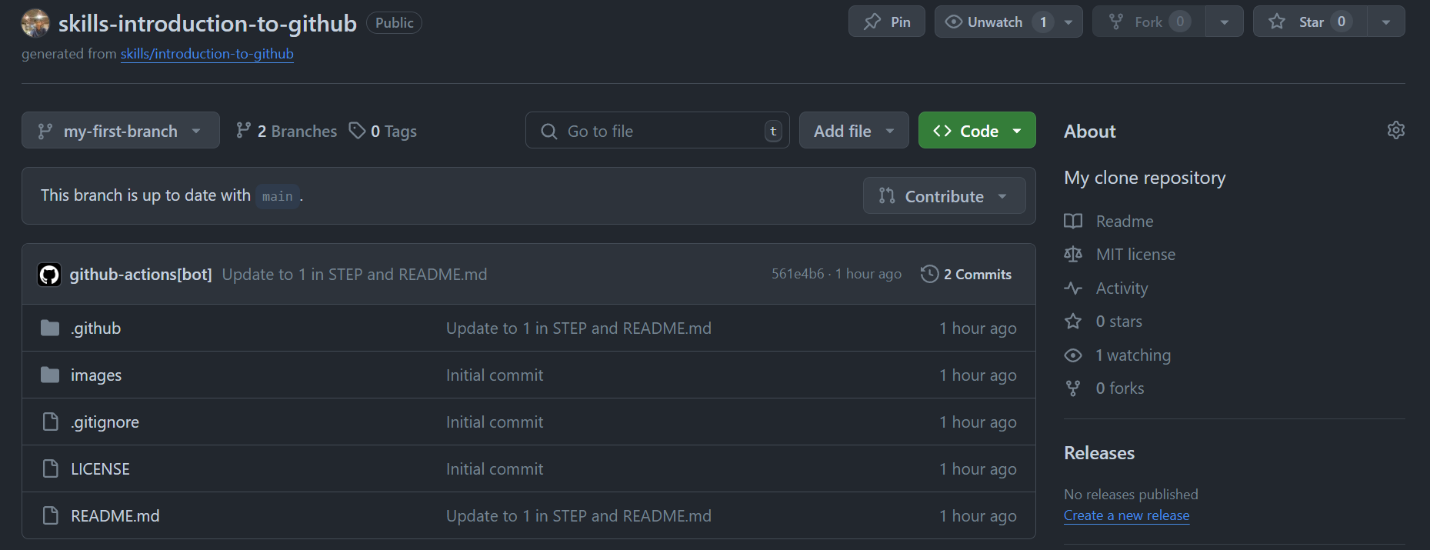
# Create a branch in the repository

**What is a branch?** A branch is a parallel version of your repository. By default, your repository has one branch named **main** and it is the definitive branch. Creating additional branches allows you to copy the **main** branch of your repository and safely make any changes without disrupting the **main** project. Many people use branches to work on specific features without affecting any other parts of the project.

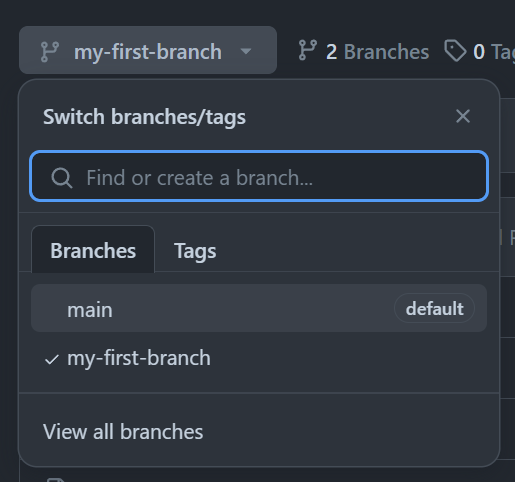
Branches allow you to separate your work from the **main** branch. In other words, everyone's work is safe while you contribute.



While in your repository click on the main tab and enter your branch name and hit the enter key or select the **create branch** option. {Type a suitable name}. You should see a screen something like this.



When creating a branch, it duplicates all the content that is in the main branch to the branch you have created. You can create as many branches as you like. To go back to your main branch just use the drop down menu.

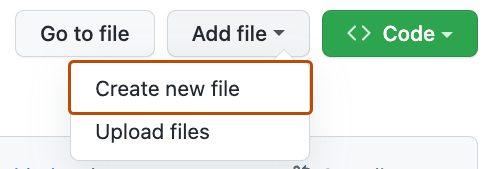


Note that any thing that is updated in the main branch will prompt a message in the other branches that it is x commits away (The x represents the number of commits in that particular branch) and anything that is updated in the other branch needs to be merged with the main branch to update the main repository.

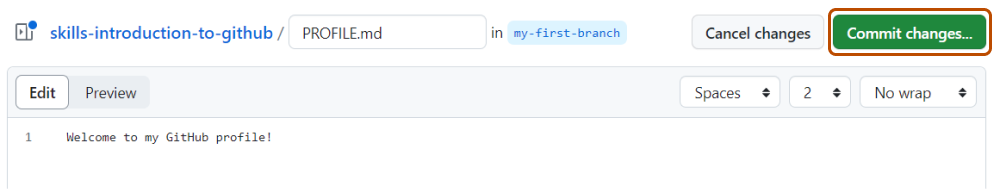
# Commit a file

**What is a commit?** A commit is a set of changes to the files and folders in your project. A commit exists in a branch and is responsible for tracking what files were uploaded, deleted or modified in the branch.

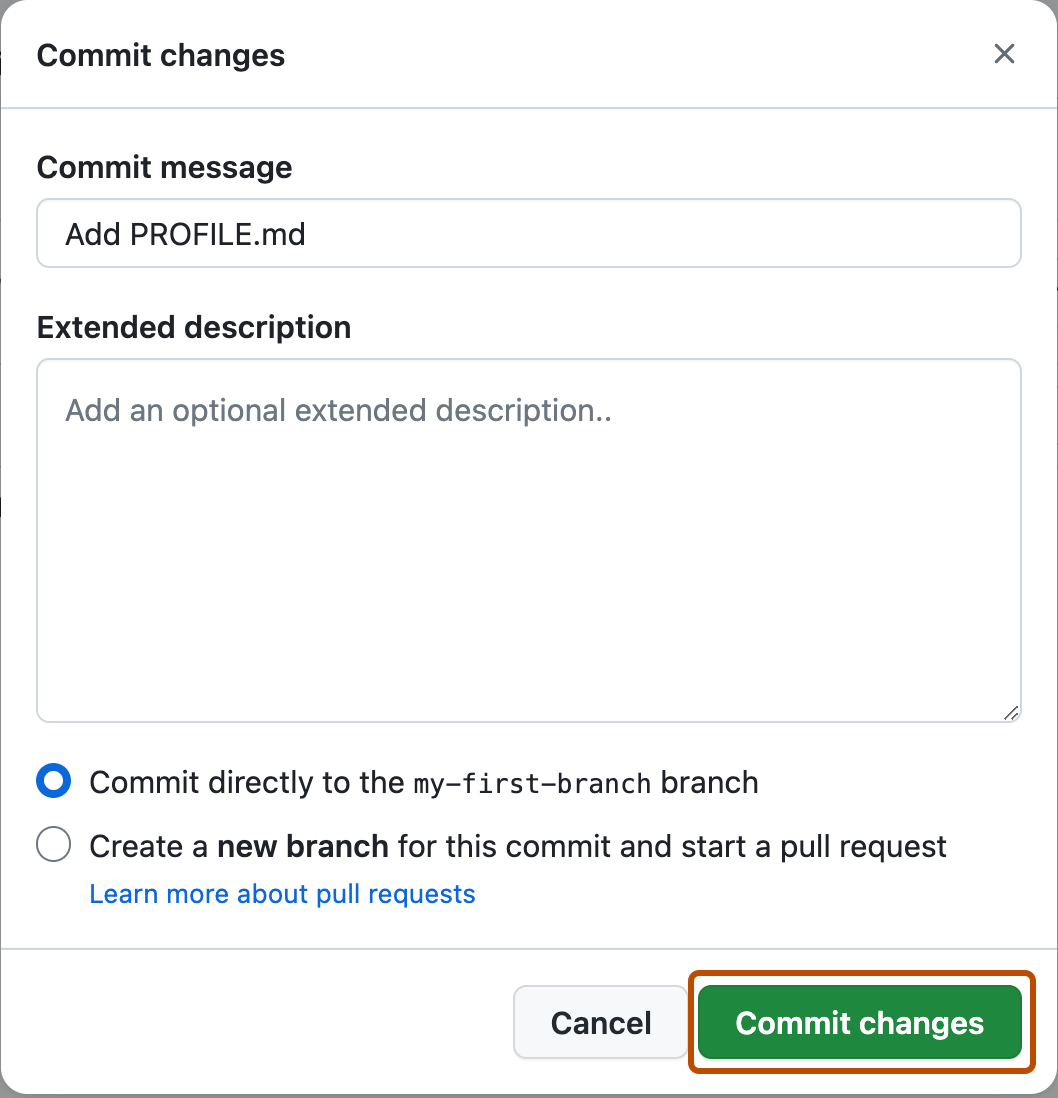
To get started On the **< > Code** tab in the header menu of your repository, make sure you're on your new branch my-first-branch.



Select the **Add file** drop-down and click **Create new file**.



In the **Name your file...** field, enter PROFILE.md.  
In the **Enter file contents here** area, Write a text i.e. Programming Fundamentals Fall 2024  
Click **Commit changes...** in the upper right corner above the contents box.  
You should see a window pop something like this down below.



For commits, you can enter a short commit message that describes what changes you made. This message helps others know what's included in your commit. GitHub offers a simple default message, but let's change it slightly for practice. First, enter Add PROFILE.md in the first text-entry field titled "Commit message". Then press the Commit changes button. We can also add an extended description for the end user to see as well as create a new branch while committing instead of the one you are trying to upload to. We can also do this for files that have been uploaded to GitHub.

1. First create a .txt file on your text (Name can be your choice).
2. Write some text in the file and save it

A black rectangular object with a black stripe

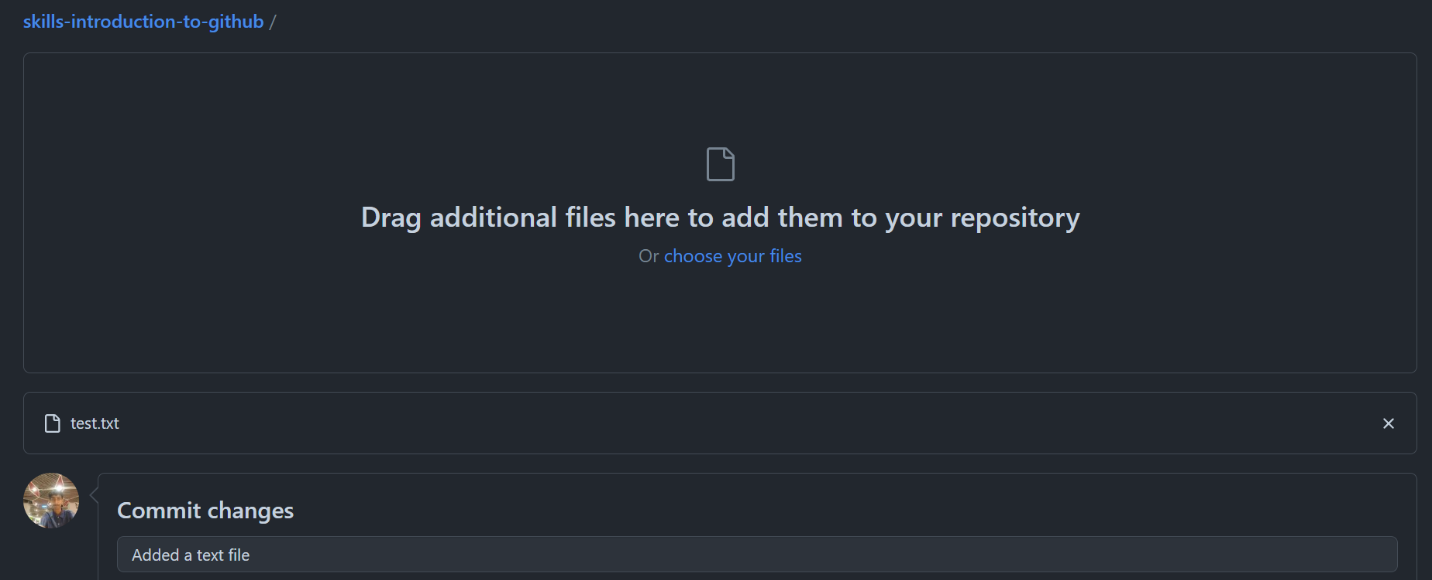
Description automatically generated

1. Then repeat the step you performed which commits a file but this time we are uploading a file to GitHub.
2. Click on the upload files option

A screenshot of a computer

Description automatically generated

1. Then insert the file or drag the file in the dialog box



1. Add some text in the Commit changes dialog box and hit the commit changes button

A screenshot of a computer

Description automatically generated

1. You should see your file in the main branch

A black background with white text

Description automatically generated

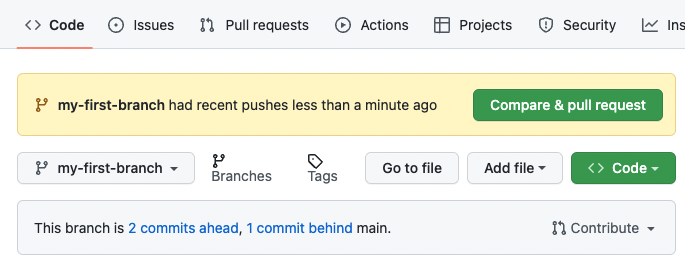
# Open a pull request

*Nice work making that commit! ✨*

Now that you have made a change to the project and created a commit, it’s time to share your proposed change through a pull request!

**What is a pull request?**: Collaboration happens on a [*pull request*](https://docs.github.com/en/get-started/quickstart/github-glossary#pull-request). The pull request shows the changes in your branch to other people and allows people to accept, reject, or suggest additional changes to your branch. In a side by side comparison, this pull request is going to keep the changes you just made on your branch and propose applying them to the main project branch.

You may have noticed after your commit that a message displayed indicating your recent push to your branch and providing a button that says **Compare & pull request**.

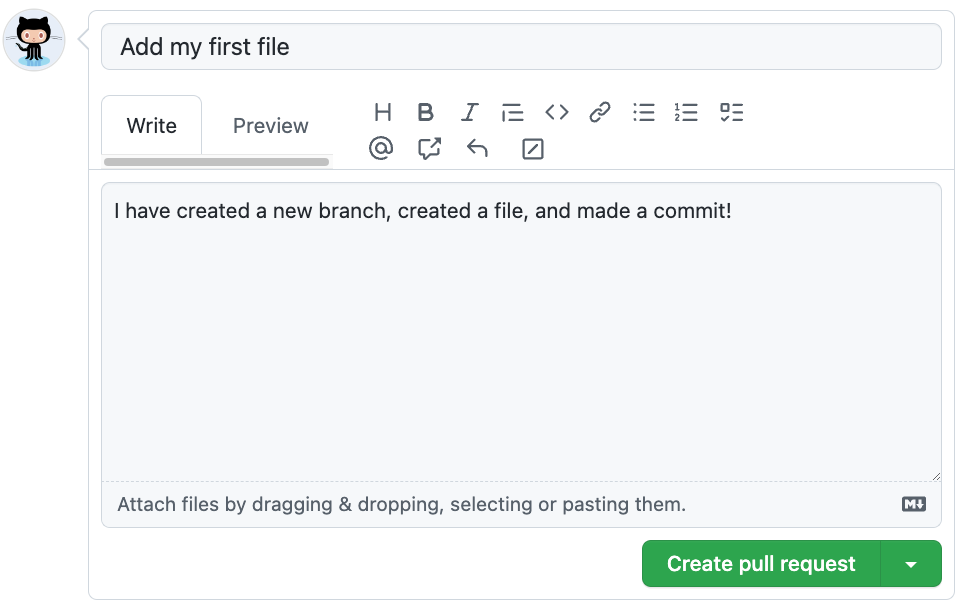


To create a pull request automatically, click **Compare & pull request**, and then skip to step 6 below. If you don't click the button, the instructions below walk you through manually setting up the pull request.

1. Click on the **Pull requests** tab in the header menu of your repository.
2. Click **New pull request**.
3. In the **base:** dropdown, make sure **main** is selected.
4. Select the **compare:** dropdown, and click my-first-branch.



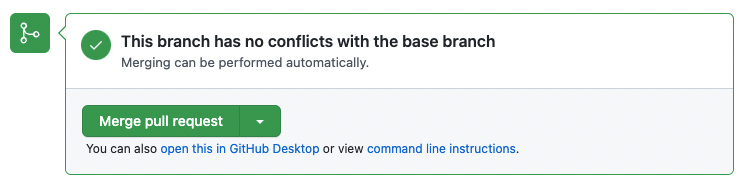
1. Click Create pull request.
2. Enter a title for your pull request. By default, the title will automatically be the name of your branch. For this exercise, let's edit the field to say Add my first file.
3. The next field helps you provide a description of the changes you made. Here, you can add a description of what you’ve accomplished so far. As a reminder, you have: created a new branch, created a file, and made a commit.

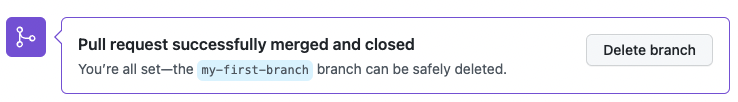


1. Click Create pull request. You will automatically be navigated to your new pull request.

# Merge a pull request

**What is a merge?**: A [*merge*](https://docs.github.com/en/get-started/quickstart/github-glossary#merge) adds the changes in your pull request and branch into the main branch. For more information about merges, see "[Merging a pull request](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/merging-a-pull-request)."

  
You would have seen a message like this. If no conflicts occur press the Merge pull request button. Click confirm. Once your branch has been merged, you don't need it anymore. To delete this branch, click **Delete branch**.

  
**Note:** deleting a branch helps to simplify your repository but keeping it is no harm as you can still reuse it for testing your work without modifying the main branch.

A screenshot of a computer

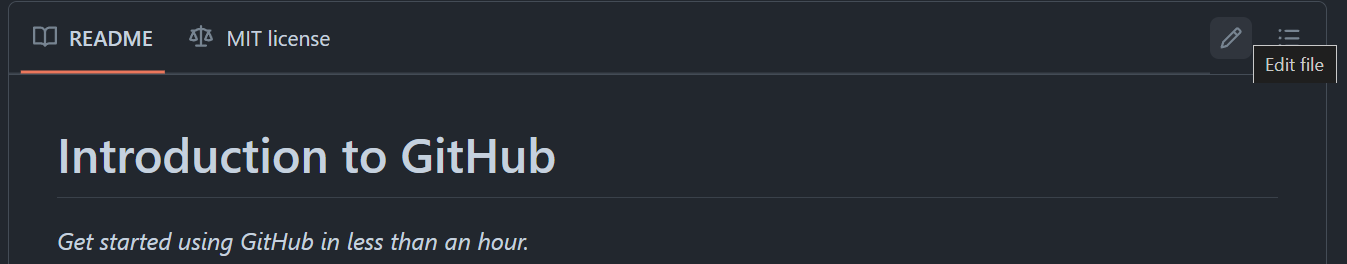
Description automatically generated

You should see that all the contents from the other branch are merged into the main branch.

# Markdown language for readme Files

Markdown is a lightweight markup language that you can use to add formatting elements to plaintext text documents. GitHub uses its own flavor of Markdown, which includes some additional features beyond the standard Markdown. Here’s a guide to some of the basic elements:

Before we start click on the pencil icon in the readme section of your GitHub Repo

  
  
Once clicked you should see a window of something like this

A screenshot of a computer

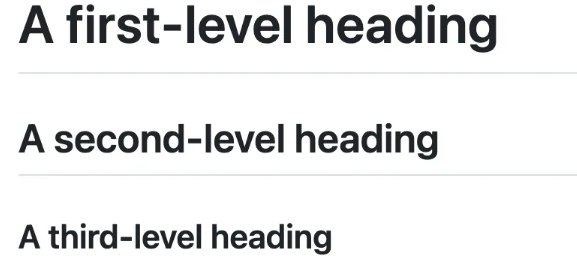
Description automatically generated

Here you are going to add changes so your readme file could look fancy and professional 😊. There is a preview page so whatever you changes you make to your file you can see the effect in realtime.

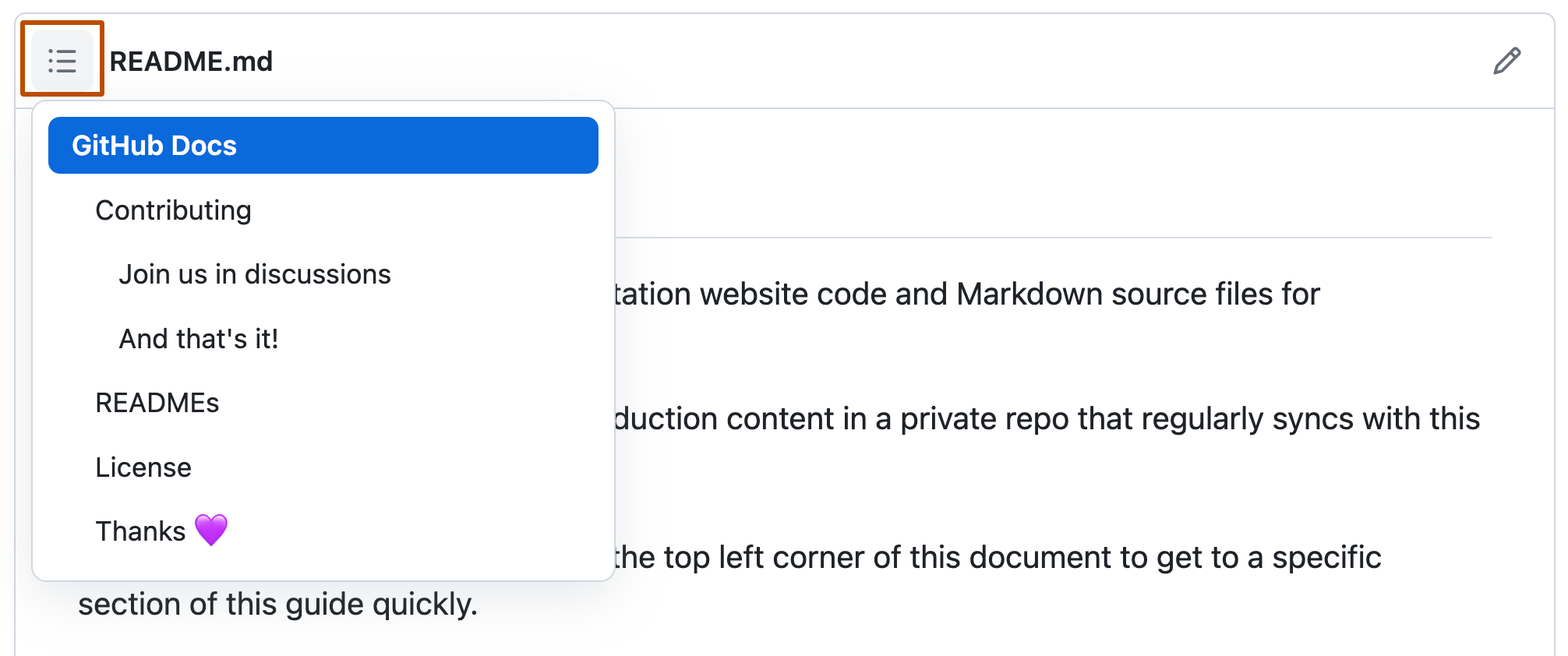
[**Headings**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#headings)

To create a heading, add one to six # symbols before your heading text. The number of # you use will determine the hierarchy level and typeface size of the heading.

1. **# A first-level heading**
2. **## A second-level heading**
3. **### A third-level heading**



When you use two or more headings, GitHub automatically generates a table of contents that you can access by clicking   within the file header. Each heading title is listed in the table of contents, and you can click a title to navigate to the selected section.



[**Styling text**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#styling-text)

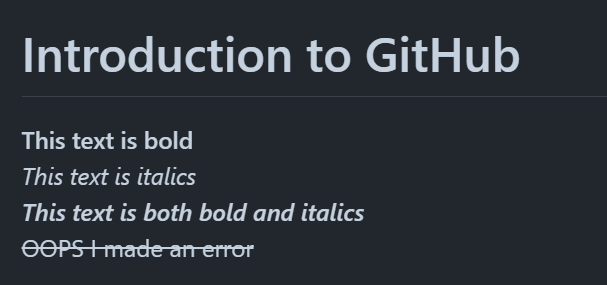
You can indicate emphasis with bold, italic, strikethrough, subscript, or superscript text in comment fields and .md files .

| **Style** | **Syntax** | **Keyboard shortcut** | **Example** | **Output** |
| --- | --- | --- | --- | --- |
| Bold | \*\* \*\* or \_\_ \_\_ | Command+B (Mac) or Ctrl+B (Windows/Linux) | \*\*This is bold text\*\* | **This is bold text** |
| Italic | \* \* or \_ \_ | Command+I (Mac) or Ctrl+I (Windows/Linux) | \_This text is italicized\_ | *This text is italicized* |
| Strikethrough | ~~ ~~ | None | ~~This was mistaken text~~ |  |
| Bold and nested italic | \*\* \*\* and \_ \_ | None | \*\*This text is \_extremely\_ important\*\* | **This text is *extremely* important** |
| All bold and italic | \*\*\* \*\*\* | None | \*\*\*All this text is important\*\*\* | ***All this text is important*** |
| Subscript | <sub> </sub> | None | This is a <sub>subscript</sub> text | This is a subscript text |
| Superscript | <sup> </sup> | None | This is a <sup>superscript</sup> text | This is a superscript text |

Example

1. # Introduction to GitHub
2. \*\*This text is bold\*\*\
3. \*This text is italics\*\
4. \*\*\*This text is both bold and italics\*\*\*\
5. ~~OOPS I made an error~~

Output

  
Note the ‘ \ ’ is used for newline. Heading do not require this.

[**Quoting text**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#quoting-text)

You can quote text with a  **>** symbol.

1. Text that is not a quote
2. > Text that is quote



[**Quoting code**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#quoting-code)

You can call out code or a command within a sentence with single backticks. The text within the backticks will not be formatted. You can also press the Command+E (Mac) or Ctrl+E (Windows/Linux) keyboard shortcut to insert the backticks for a code block within a line of Markdown.

1. Use `git status` to list all new or modified files that haven't yet been committed.
2. Some basic Git commands are:
3. ```
4. git status
5. git add
6. git commit
7. **```**

A screen shot of a computer

Description automatically generated

[**Lists**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#lists)

You can make an unordered list by preceding one or more lines of text with -, \*, or +.

1. - George Washington
2. \* John Adams
3. + Thomas Jefferson



To order your list, precede each line with a number.

1. 1. James Madison
2. 2. James Monroe
3. 3. John Quincy Adams



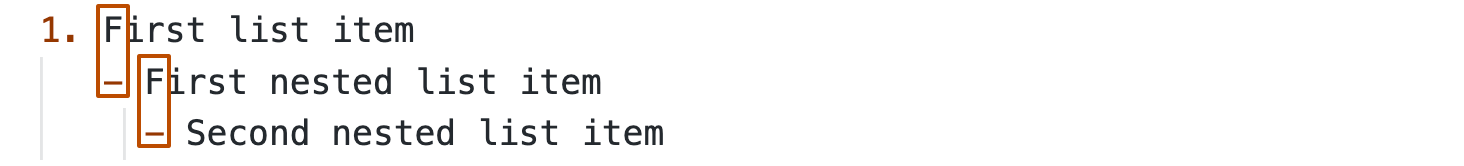
[**Nested Lists**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#nested-lists)

You can create a nested list by indenting one or more list items below another item.

To create a nested list using the web editor on GitHub or a text editor that uses a monospaced font, like [Visual Studio Code](https://code.visualstudio.com/), you can align your list visually. Type space characters in front of your nested list item until the list marker character (- or \*) lies directly below the first character of the text in the item above it.

1. 1. First list item
2. - First nested list item
3. - Second nested list item

Output

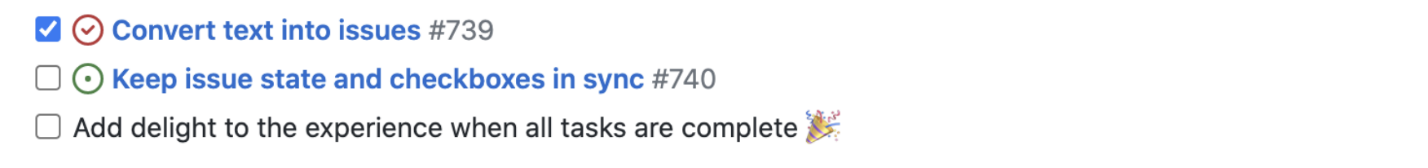




[**Task lists**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#task-lists)

To create a task list, preface list items with a hyphen and space followed by [ ]. To mark a task as complete, use [x].

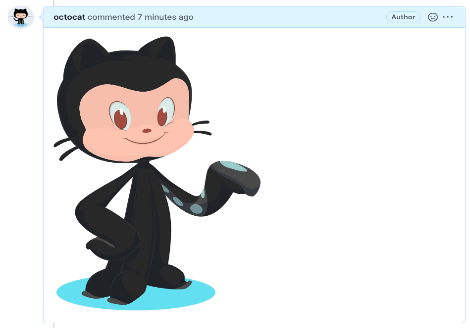
1. - [x] #739
2. - [ ] https://github.com/octo-org/octo-repo/issues/740
3. - [ ] Add delight to the experience when all tasks are complete :tada:



[**Images**](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#images)

You can display an image by adding ! and wrapping the alt text in [ ]. Alt text is a short text equivalent of the information in the image. Then, wrap the link for the image in parentheses ().

1. ![Screenshot of a comment on a GitHub issue showing an image, added in the Markdown, of an Octocat smiling and raising a tentacle.](<https://myoctocat.com/assets/images/base-octocat.svg>)



Note that direct weblink and link to your repository path where the image is located can be utilized in this manner.

**Now What is Git?**

Git is a very popular version control system.

**What is a Version Control System (VCS)?**

A version control system records the changes made to code, over time, in a database. This database is referred to as the repository. The project history can be viewed here and we can also see who has made what changes, why the changes were made, and when the changes were made. If the project runs into any issues after the changes, it can be reverted back to an older, more stable version.

Without a version control system, a lot of manual work will be needed to maintain versions of the project. This would become even more difficult to manage if multiple people are working on the project on their own PCs.

The two main benefits of using any VCS are:

* Easier to track project history
* Easier to collaborate on the project

You can have two types of VCS:

* Distributed
* Centralized

**Distributed Version Control Systems:**

In a distributed VCS, all team members have a copy of their project on their machine. Therefore, each member can save changes to the projects locally.

Git is an example of a Distributed VCS

**Centralized Version Control Systems:**

A central server is present, and all team members have to connect to the central server to get the latest copy of the project code. They then update their changes as a new version on the central server.

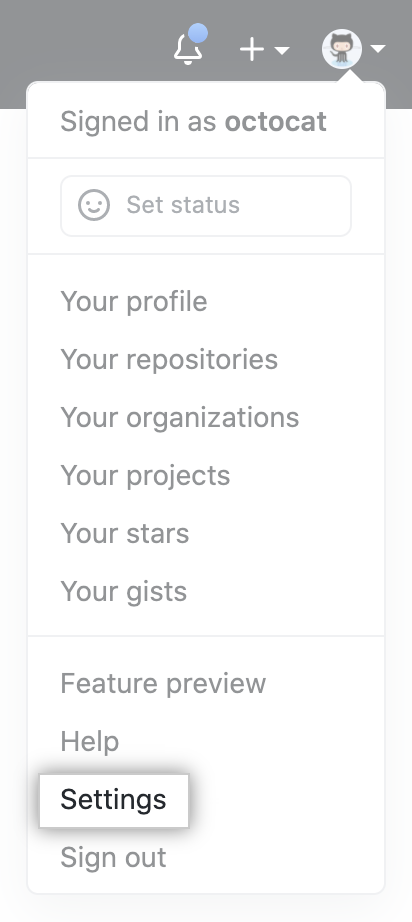
The disadvantage of a centralized server is the single point of failure. In case of server failure, the team members cannot update their changes to the server, nor can they obtain the recent copy of the project.

Now let’s try to install, setup and use Git! Follow the following steps

1. Download and install the latest version of Git from <https://git-scm.com/downloads>
2. Make sure it is installed properly in your system by using the following command in your command prompt:

git --version



1. Open Git Bash.
2. Set a Git username using the following command:

git config --global user.name "Your Name"

1. Confirm that you have set the Git username correctly:

git config --global user.name

1. Set an email address in Git using the following command:

git config --global user.email "email@example.com"

1. Confirm that you have set the email address correctly:

git config --global user.email

1. Add the email address to your account on GitHub, so that your commits

are attributed to you and appear in your contributions graph.

We will need this later. On your GitHub account:

* 1. In the upper-right corner of any page, click your profile photo,

then click **Settings**.

* 1. In the "Access" section of the sidebar, click Emails.
  2. In "Add email address", type your email address and click **Add**.
  3. Verify your email address.
  4. In the "Primary email address" list, select the email address

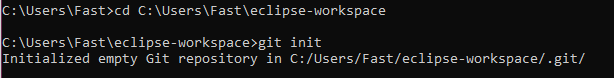
you'd like to associate with your web-based Git operations.

1. In the command prompt (or Git Bash) browse to the directory that you wish the create the repository in. You can do this by typing:

cd <directory path here>

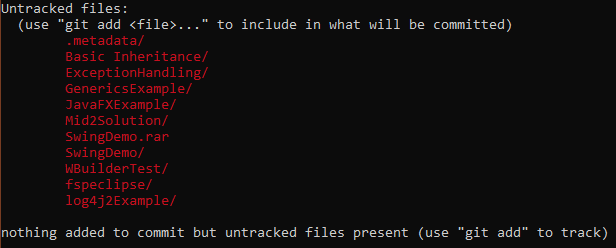
1. Type the following to create a repository in the location:

git init



1. You can check the status of your repository by using the following command:

git status



1. Before we can commit files to Git, we must first stage them. In order to stage the files, we use the following command:

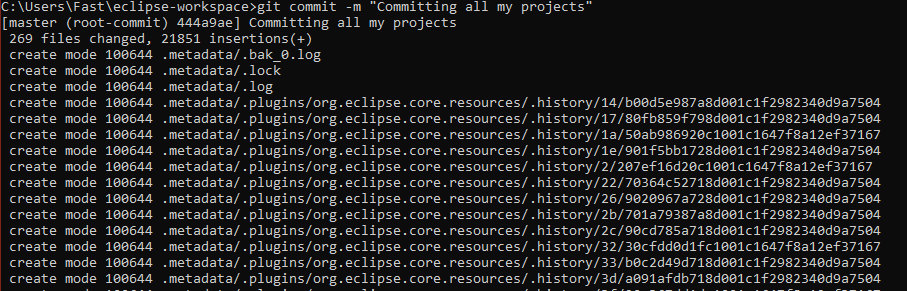
git add <name of file/folder you want to add>



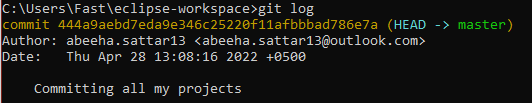
Note: If your folder has whitespace, you will get an error, you will need to wrap it in single quotes ('). For example: git add 'folder with space/'.

1. You can use git add . to add all files and folders to the staging area.
2. A **commit** is a snapshot of our code at a particular time, which we are saving to the commit history of our repository.

git commit -m "Commit message"



1. To view the commit, you can use the command: git log



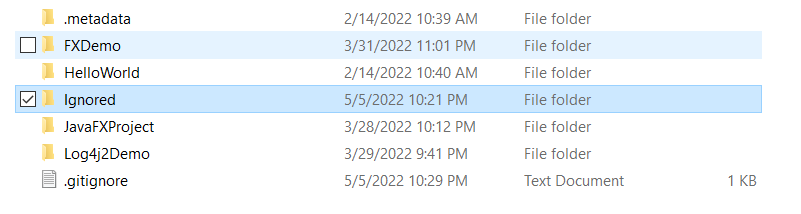
1. You can also set it up so that you can ignore some files. First of all, navigate to the location of your repository (which you should be in currently, if you have followed the steps accurately). Type the following to create a .gitignore file. If it is successful, there will be no message.

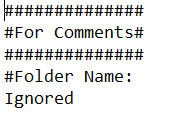
touch .gitignore

1. If you want to ignore a file that is already checked in, you must untrack the file before you add a rule to ignore it. From your terminal, untrack the file using the following command.

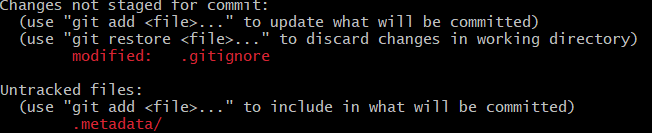
git rm –cached -r FILENAME

1. Add the files and folders that you want to ignore inside the .gitignore file.





1. Use the git status command to check on the status of your repository.



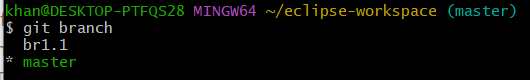
1. Commit the updated gitignore file.
2. When we initialize a repository and start making commits, they are saved to the master branch by default. We can create a branch using the following command:

git branch <new-branch-name>

There will be no message if it is successful.

1. To display a list of all the branches, you can use the following command:

git branch



1. To switch branches, you have to “checkout” a branch by using the following command:

git checkout <branch-name>



1. To create a new branch and change to it at the same time, you can use the -b flag:

git checkout -b <branch-name>



1. To merge the changes from a different branch into your current branch, you can use this command:

git merge <branch-name>



1. Finally, if you want to delete a branch, you can use this command:

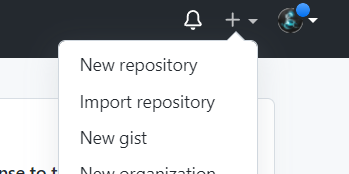
git branch -d <branch-name>



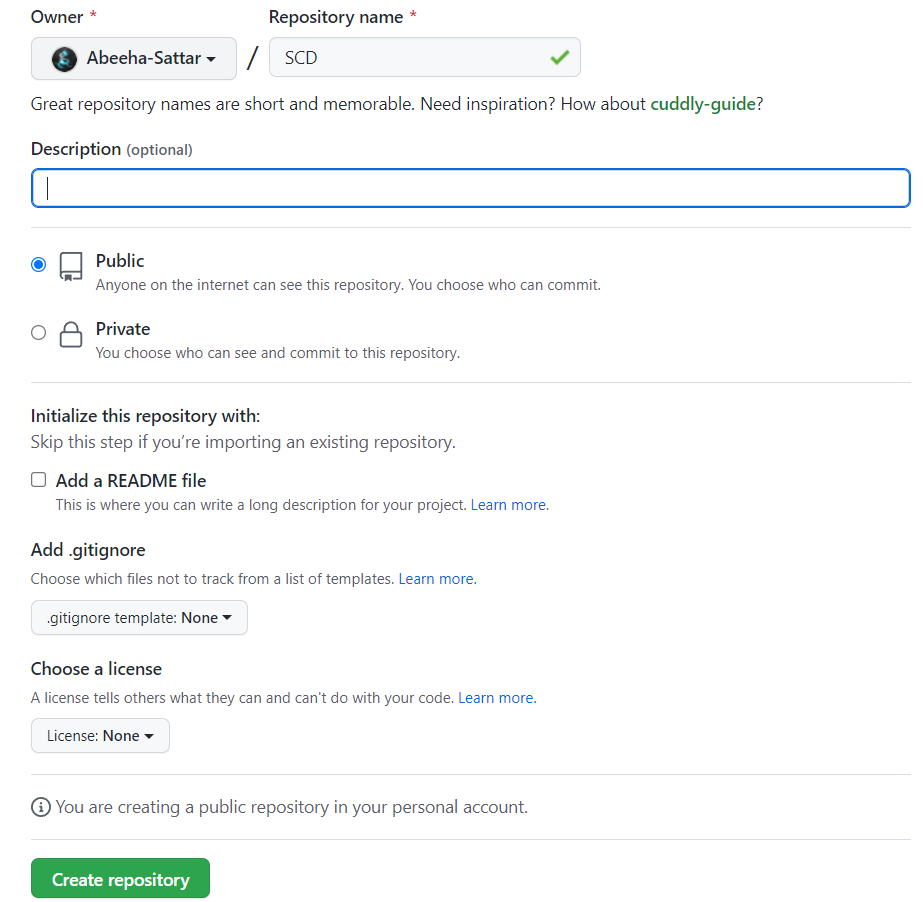
**GitHub**

GitHub provides a web based interface for git. It also provides an online server where you can store the files. (Remember that Git stores the files locally and not on a server)

First of all, create an account on GitHub, and proceed with the following steps:



1. Create a repository by clicking on the plus icon:
2. Fill out the repository name (do not use spaces), and create a repository with default options:



1. After you have created your repository, let’s push some changes to it. First of all we have create an origin to which we can push to. Use the following lines to create an origin:

git remote add origin <https://github.com/Abeeha-Sattar/SCD.git>

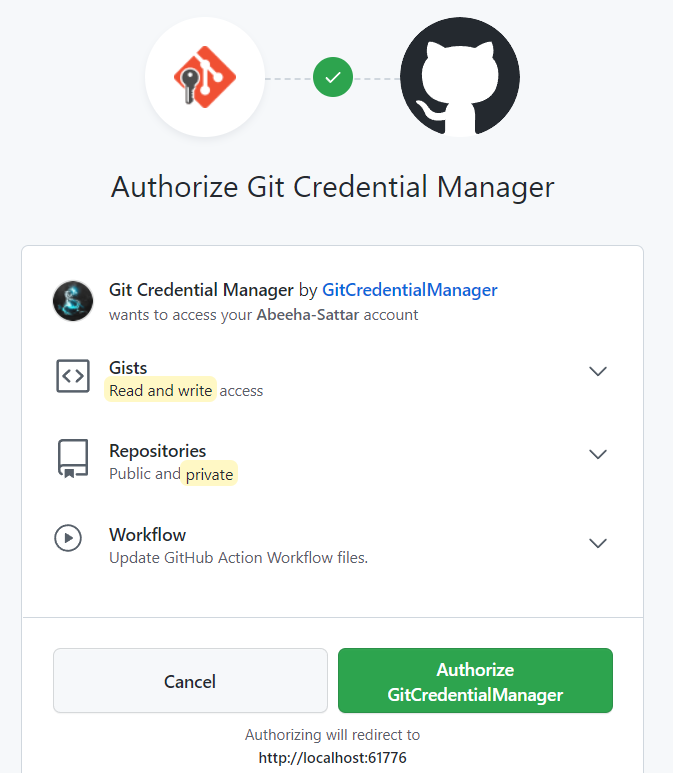


1. To push to origin, use the following command:

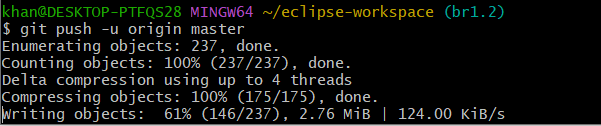
git push -u origin <branchname>



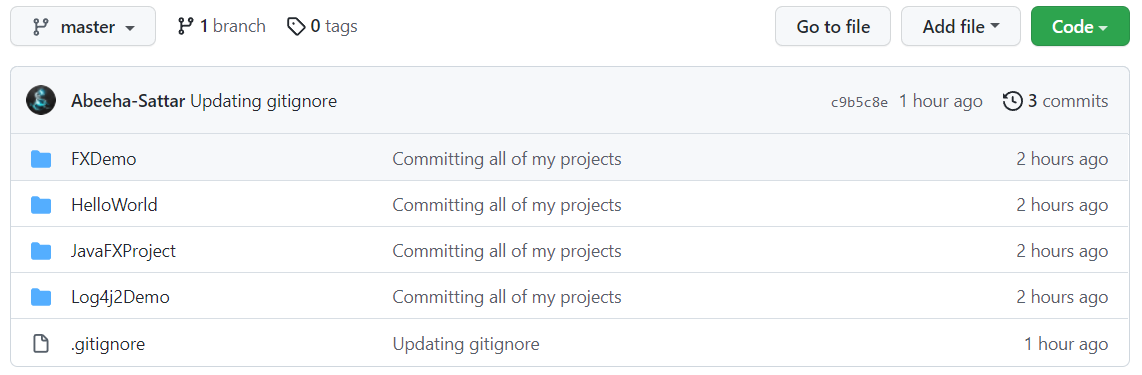
1. You will be asked to authorize the login. Authorize via browser.



1. You will be able to view the progress on your git bash/cmd



1. Refresh your repository page after you have pushed your files successfully.

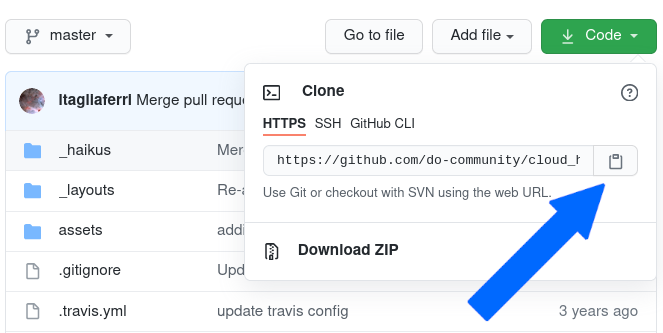


1. Create a new branch where you will get the recent copy of the project from the repository – this is called cloning (using bash or cmd)

git checkout -b <branch-name>

1. Clone the repository using the following command:

git clone https://github.com/your-username/repository.git



1. You can view the progress as it is being cloned:

