**INDEX**

1. Git and GitHub 01
   1. Git and Github are different 01
2. Terminology 02
   1. Repository 02
   2. Your config setting 03
   3. Creating a repository 04
   4. Commit Way 05
   5. Stage 06
   6. Commit Command 07
   7. Logs 08
   8. Gitignore 09
3. Branched in git 11
   1. Head in git 11
   2. Git Graph tool 14
   3. Merging branches 14
      1. Fast Forward Merge 14
      2. Not Fast Forward Merge 17
   4. Manging conflicts 18
   5. Conflicts creation and solving 19
   6. Rename a branch 22
   7. Delete a branch 22
   8. Checkout a branch 23
   9. List all branches 23
4. Diff stash and tags 24
   1. Git diff 24
      1. Comparing working Directory and Staging Area 24
      2. Comparing Staging Area with Repository 25
      3. Comparing Between Branches 25
      4. Comparing Specific Commits 25
   2. Git Stash 26
      1. Naming the stash 26
      2. View the Stash list 26
      3. Apply the stash 27
      4. Apply the Specific Stash 27
      5. Applying and Dropping the Stash 27
      6. Drop the Stash 27
      7. Applying Stash to a Specific branch 28
      8. Clearing the Stash 28
   3. Git Tags 28
      1. Creating a Tag 28
      2. Creating an annotated tag 29
      3. List all tags 29
      4. Tagging a specific Commit 29
      5. Push tags to remote repository 29
      6. Delete a tag 30
      7. Delete tag on remote repository 30
5. Rebase and reflog 31
   1. Rabase in git 31
   2. Merge commits 31
   3. Rebase 32
   4. Ensure you are on the branch you want to rebase 32
   5. Resolve any conflicts 33
   6. Git reflog 33
   7. View the reflog 33
   8. Find specific commit 33
   9. Recover lost commit or changes 34
6. Started with Github 35
   1. Configure your config file 35
   2. Setup ssh key and add to github 35
      1. Step 1: Generate a new SSH Ke5 36
      2. Save the Key 36
      3. Add key to your ssh-agent 36
      4. Add key to github 36
   3. Adding code to remote repository 36
      1. Check remote url setting 37
      2. Add remote repository 37
      3. Push code to remote repository 38
      4. Setup an upstream remote 38
   4. Readme.md file 39
   5. Get code from remote repository 40
      1. Fetch Code 40
      2. Pull Code 41
7. Open Source 42

**1. Git and GitbHub**

**Git** is a version control system that allows you to track changes to your files and collaborate with others. It is used to manage the history of your code and to merge changes from different branches. I can understand that as of now these terms like version control, branches, and merges are not familiar to you.

**1.1.** **Git and Github are different**

Git is a version control system that is used to track changes to your files. It is a free and open-source software that is available for Windows, macOS, and Linux. Remember, GIT is a software and can be installed on your computer.

Github is a web-based hosting service for Git repositories. Github is an online platform that allows you to store and share your code with others. It is a popular platform for developers to collaborate on projects and to share code. It is not that Github is the only provider of Git repositories, but it is one of the most popular ones.

ID and PASS



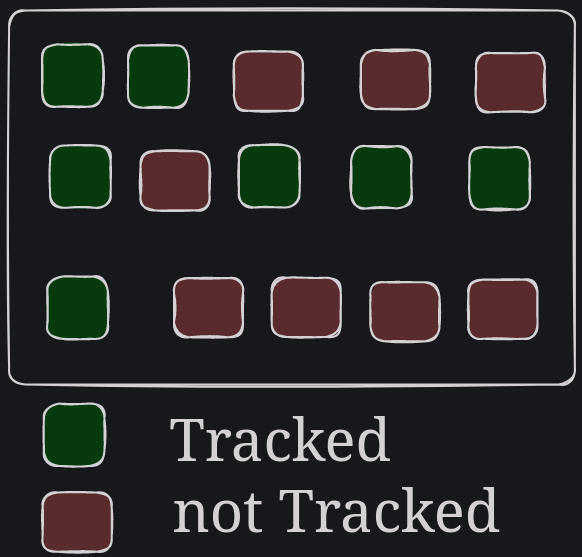
**2. Terminology**

**2.1. Repository**

A repository is a collection of files and directories that are stored together. It is a way to store and manage your code. A repository is like a folder on your computer, but it is more than just a folder. It can contain other files, folders, and even other repositories. You can think of a repository as a container that holds all your code.

There is a difference between a software on your system vs tracking a particular folder on your system. At any point you can run the following command to see the current state of your repository:

git status

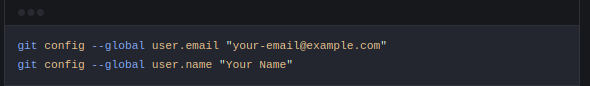


Not all folders are meant to be tracked by git. Here we can see that all green folders are projects are getting tracked by git but red ones are not.

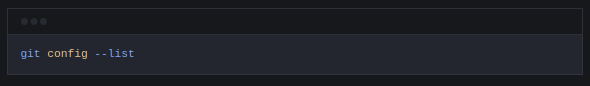
**2.2. Your config settings**

Github has a lot of settings that you can change. You can change your username, email, and other settings. Whenever you checkpoint your changes, git will add some information about your such as your username and email to the commit. There is a git config file that stores all the settings that you have changed. You can make settings like what editor you would like to use etc. There are some global settings and some repository specific settings.

Let’s setup your email and username in this config file.



Now you can check your config settings:



This will show you all the settings that you have changed.

Example:

**2.3. Creating a repository**

For creating repository first you need to check the status if you get it like this

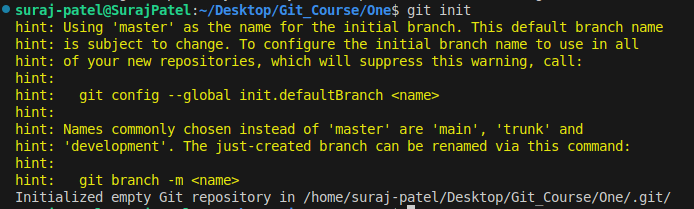
git status



fatal: not a git repository (or any of the parent directories): .git

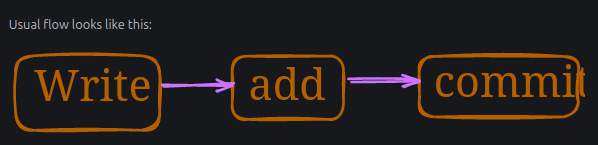
then creating repository

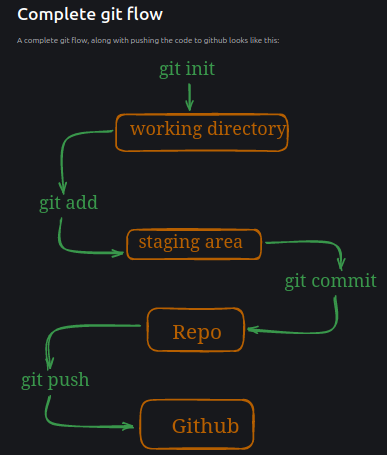
git init is used to create repository



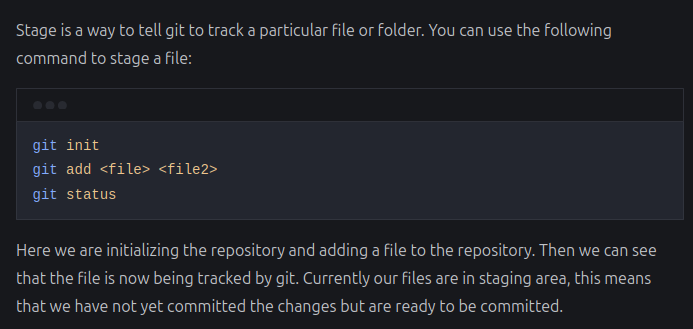
**2.4. Commit Way**

commit is a way to save your changes to your repository.



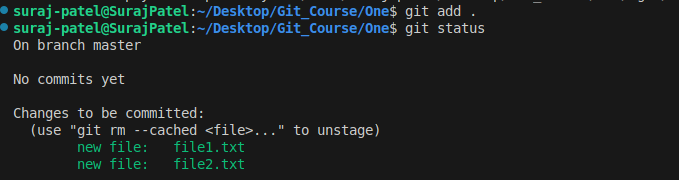


**2.5. Stage**

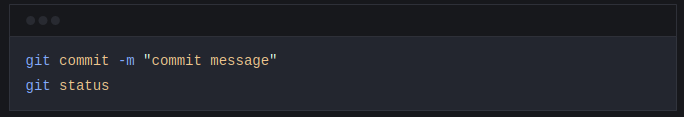
Here git add <file1> <file2> to add is used to stage a file to commit it

Or other way

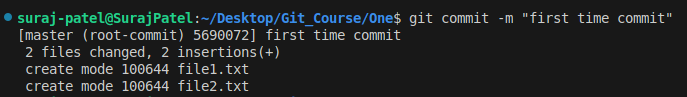
git add . To add all file

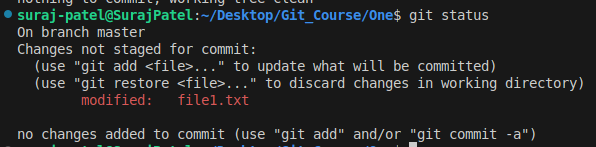
And for remove a file you need to write code git rm –cached<file>...” to unstage

**2.6. Commit**

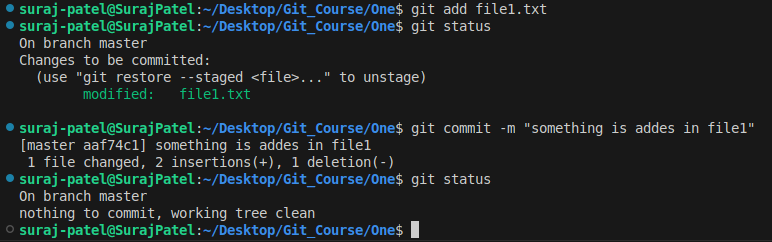
Here we are committing the changes to the repository. We can see that the changes are now committed to the repository. The -m flag is used to add a message to the commit. This message is a short description of the changes that were made. You can use this message to remember what the changes were. Missing the -m flag will result in an action that opens your default settings editor, which is usually VIM. We will change this to vscode in the next section.

Example

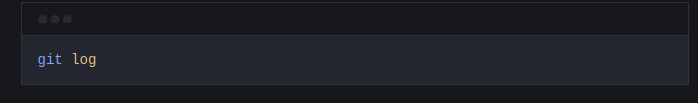
Now in the file1 we added something and now checking with status



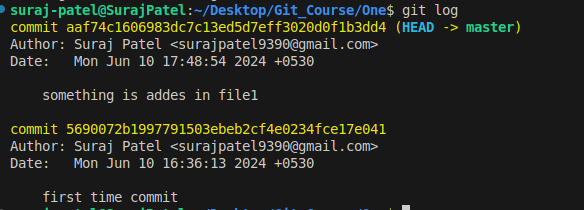
Now we commit file1 only to save in git



**2.6. Logs**

This command will show you the history of your repository. It will show you all the commits that were made to the repository. You can use the --oneline flag to show only the commit message. This will make the output more compact and easier to read.

Example

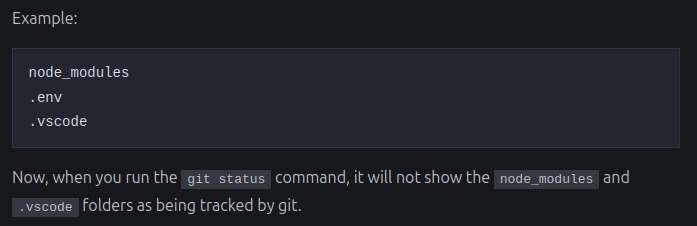


Now using oneline command to show log



**2.7. Gitignore**

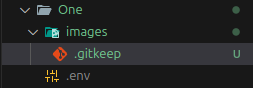
Gitignore is a file that tells git which files and folders to ignore. It is a way to prevent git from tracking certain files or folders. You can create a gitignore file and add list of files and folders to ignore by using the following command:



git is ignored the .env file because of .gitignore file

**Blank Folder**

By defaults blanks folder get ignore by git to keep blank folder we use .gitkeep



3. Branched in git

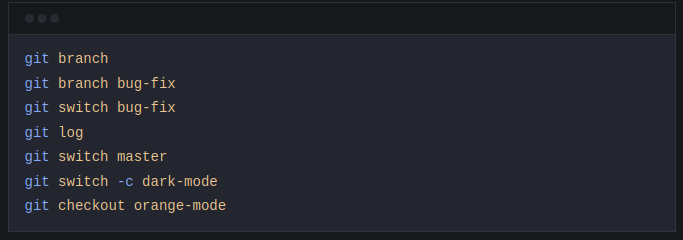
Branches are a way to work on different versions of a project at the same time. They allow you to create a separate line of development that can be worked on independently of the main branch. This can be useful when you want to make changes to a project without affecting the main branch or when you want to work on a new feature or bug fix.

Some developers can work on Header, some can work on Footer, some can work on Content, and some can work on Layout. This is a good example of how branches can be used in git.

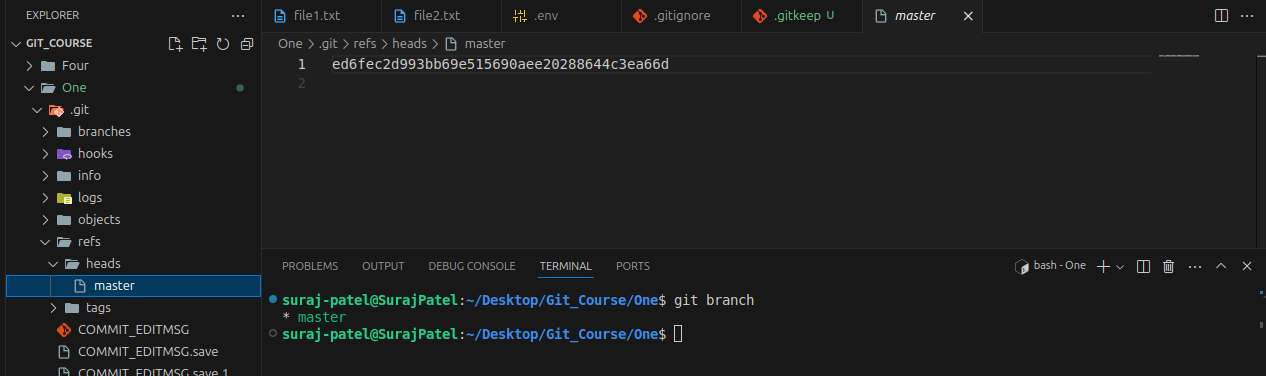
**3.1. HEAD in git**

The HEAD is a pointer to the current branch that you are working on. It points to the latest commit in the current branch. When you create a new branch, it is automatically set as the HEAD of that branch.

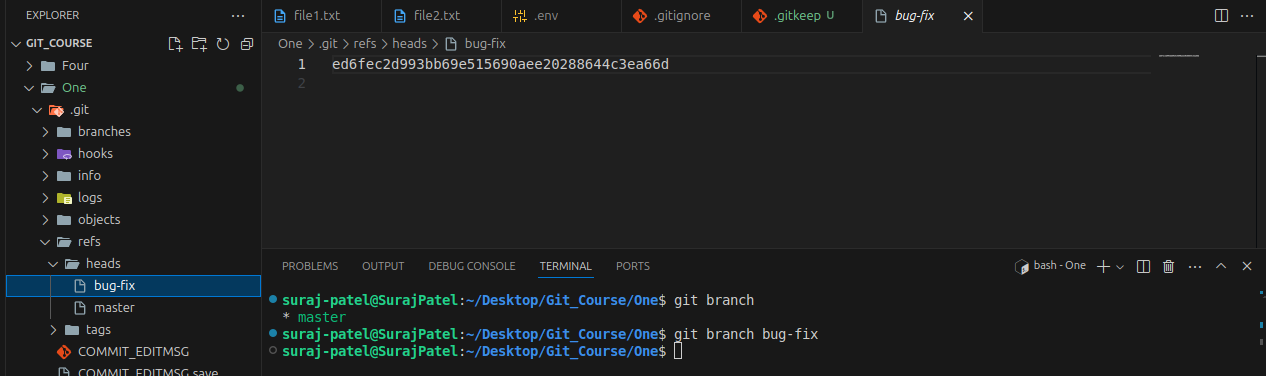
Creating a new branch

Some points to note:

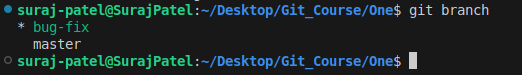
* git branch (gives all branch names) - This command lists all the branches in the current repository. Example

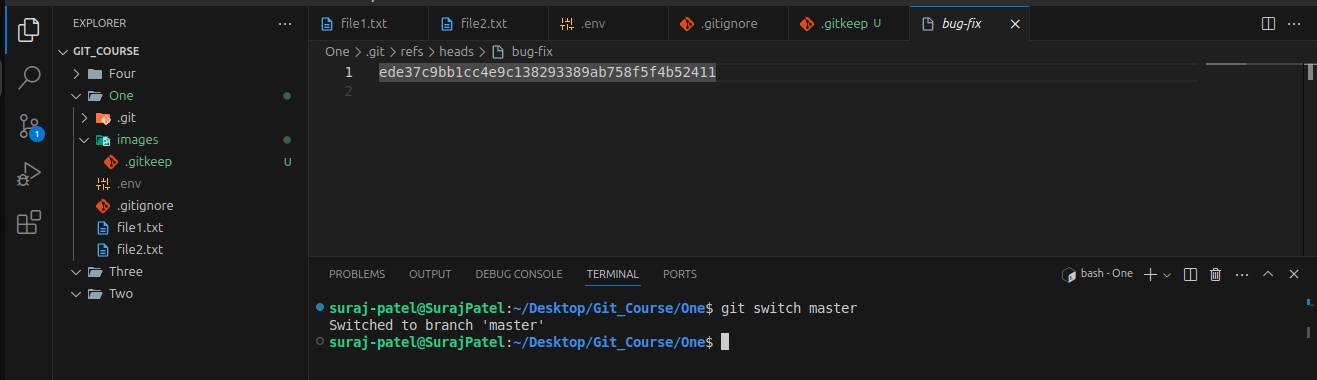


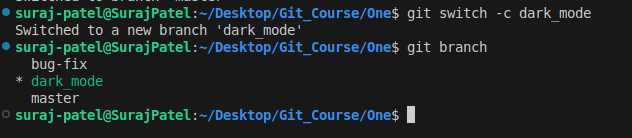
* git branch bug-fix(Creating New Branch) - This command creates a new branch called bug-fix. Example:

But write now selected branch is master if you want to work on created branch you need to switch the branch

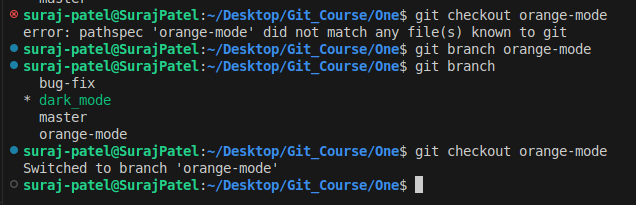
* git switch bug-fix (Switch the branch) - This command switches to the bug-fix branch. Example

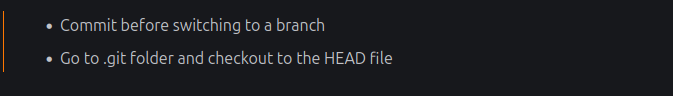


* git log - This command shows the commit history for the current branch.
* git switch master (Switch to original path then what happened) - This command switches to the master branch. BugFix file not showing because this file in another branch and branched name is (bug-fix)
* git switch -c dark-mode (create if exited and switch)- This command creates a new branch called dark-mode. the -c flag is used to create a new branch.

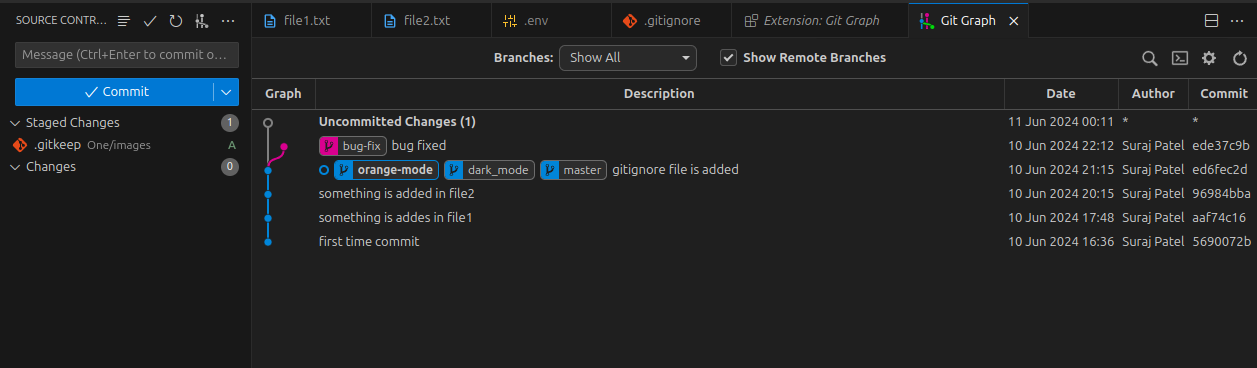
It is used to create branch and switch when branch is created and if branch is exited then it only switch

* git checkout orange-mode (It is used to switch and most used)- This command switches to the orange-mode branch.

It is used to swtich if exited and also tell that if branch is exited or not



**3.2. Git Graph tool**

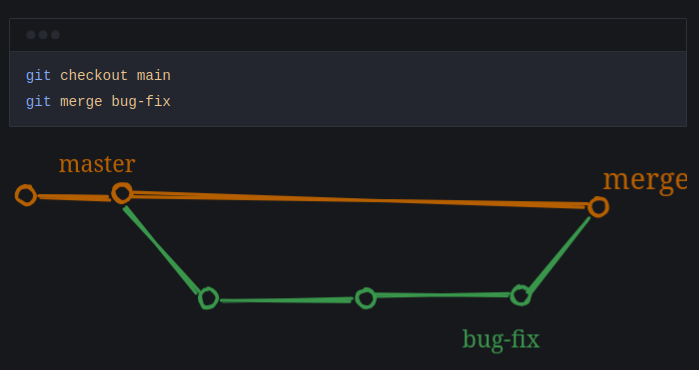


**3.3. Merging branches**

**3.3.1. Fast-forward merge**

This one is easy as branch that you are trying to merge is usually ahead and there are no conflicts.

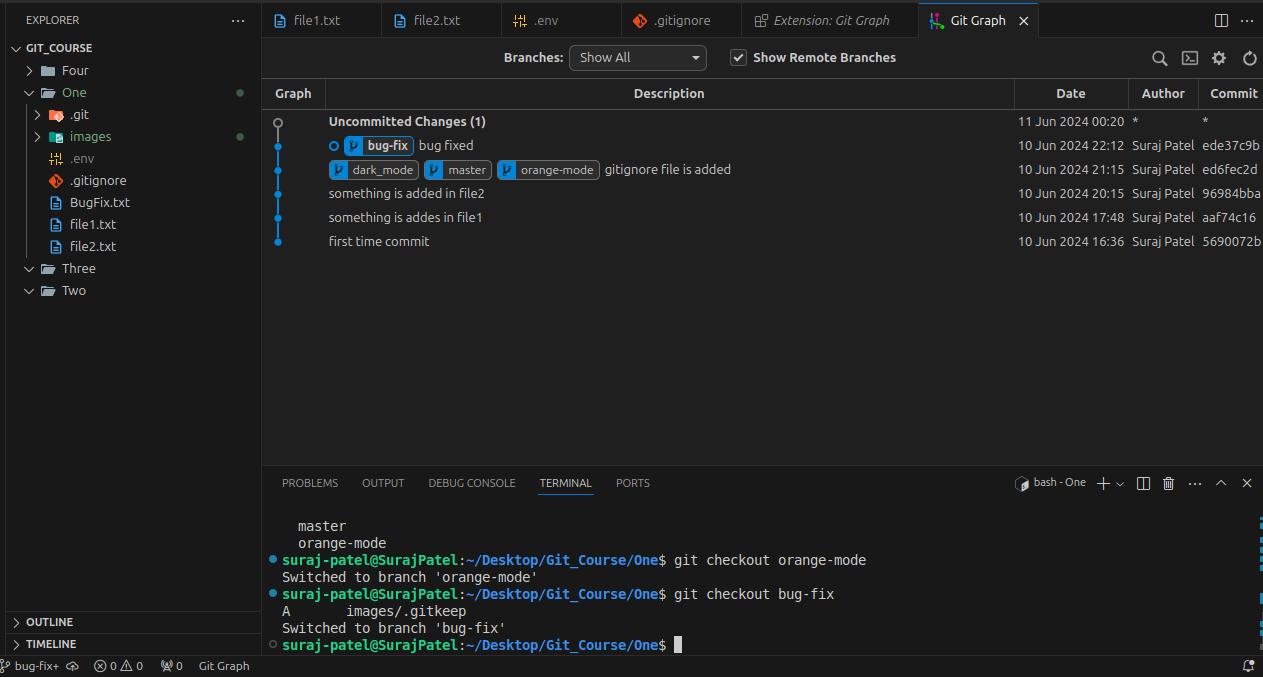
When you are done working on a branch, you can merge it back into the main branch. This is done using the following command:

Some points to note:

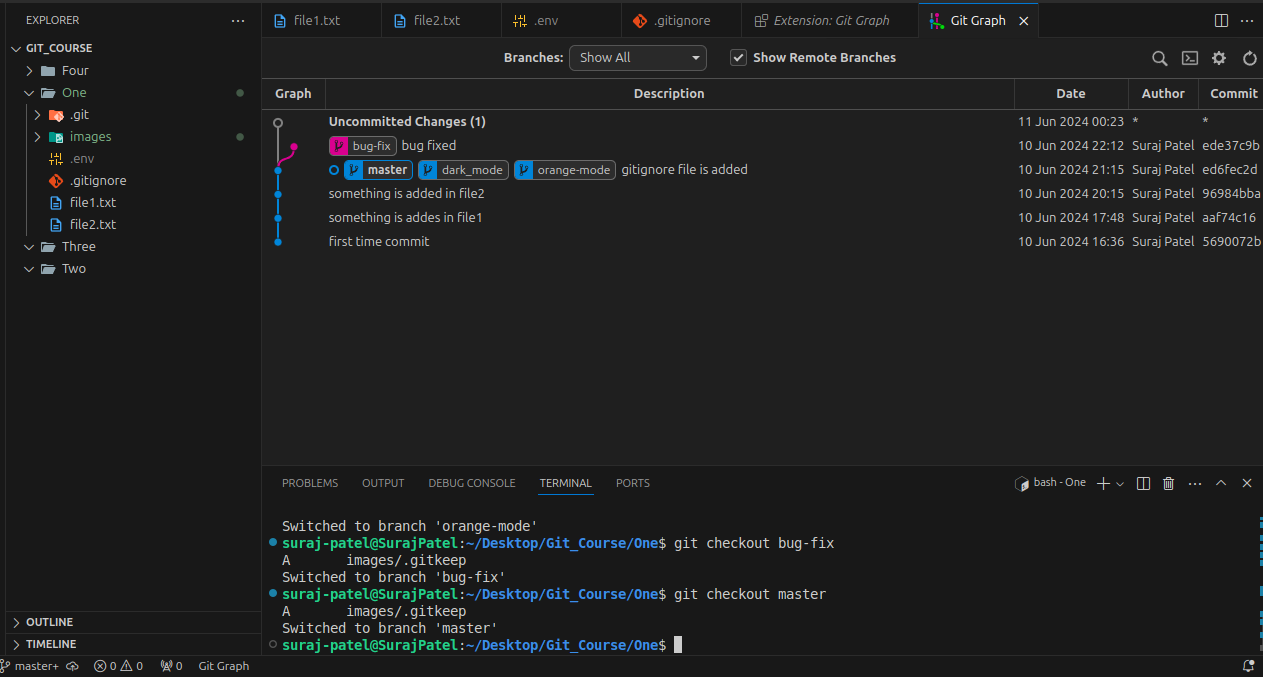
* git checkout main - This command switches to the main branch.
* git merge bug-fix - This command merges the bug-fix branch into the main branch.

This is a fast-forward merge. It means that the commits in the bug-fix branch are directly merged into the main branch. This can be useful when you want to merge a branch that has already been pushed to the remote repository.

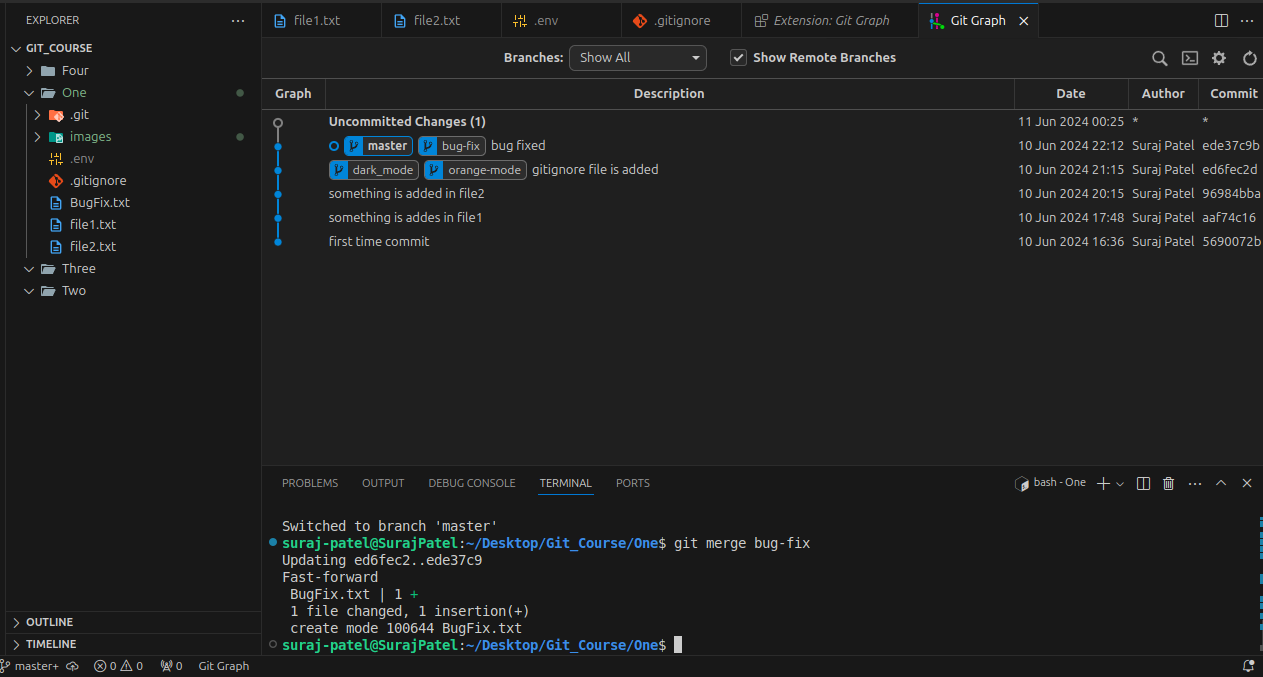
Example:



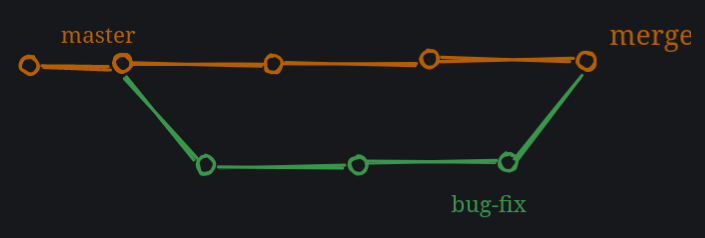
* In branch bug-fix you can see file BugFix but not in master branch(main) see below image

For merge file then BugFix file is also present in master

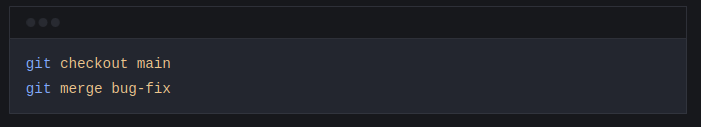
git merge bug-fix



**3.3.2. Not fast-forward merge**

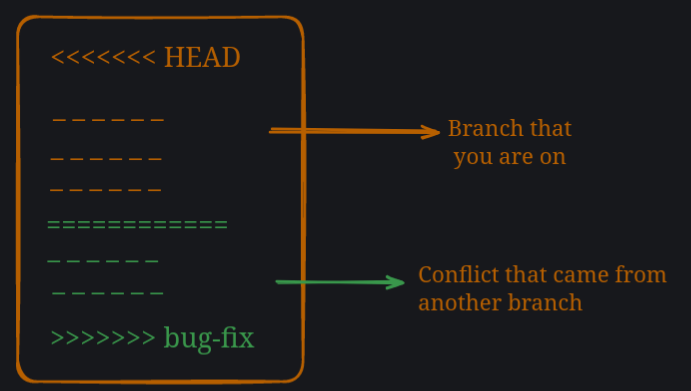
In this type of merge, the master branch also worked and have some commits that are not in the bug-fix branch. This is a not fast-forward merge.

When you are done working on a branch, you can merge it back into the main branch. This is done using the following command:



If the command are same, what is the difference between fast-forward and not fast-forward merge?

The difference is resolving the conflicts. In a fast-forward merge, there are no conflicts. But in a not fast-forward merge, there are conflicts, and there are no shortcuts to resolve them. You have to manually resolve the conflicts. Decide, what to keep and what to discard. VSCode has a built-in merge tool that can help you resolve the conflicts.

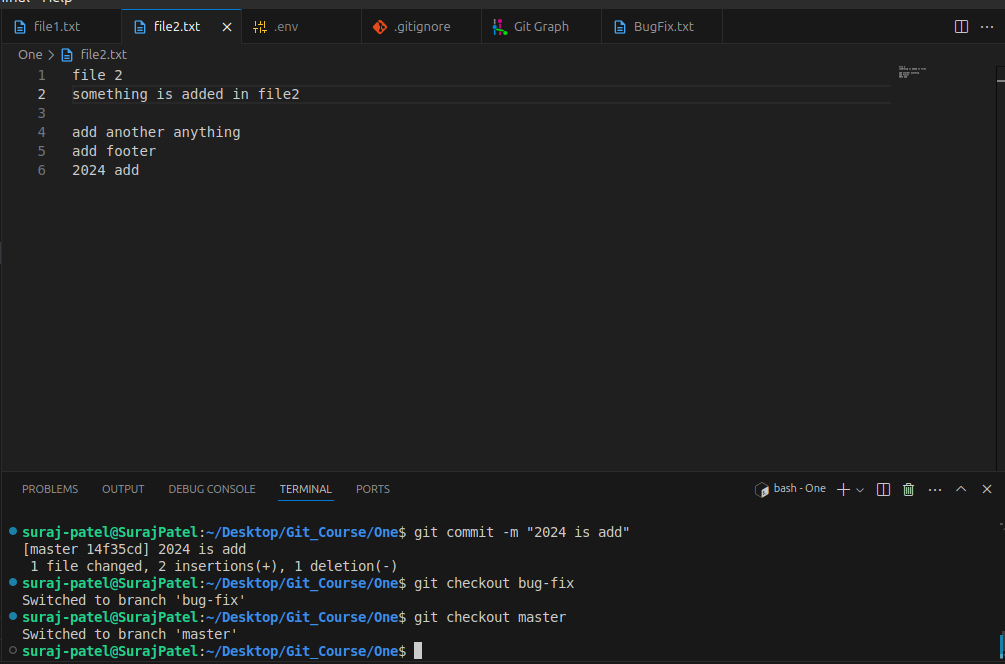


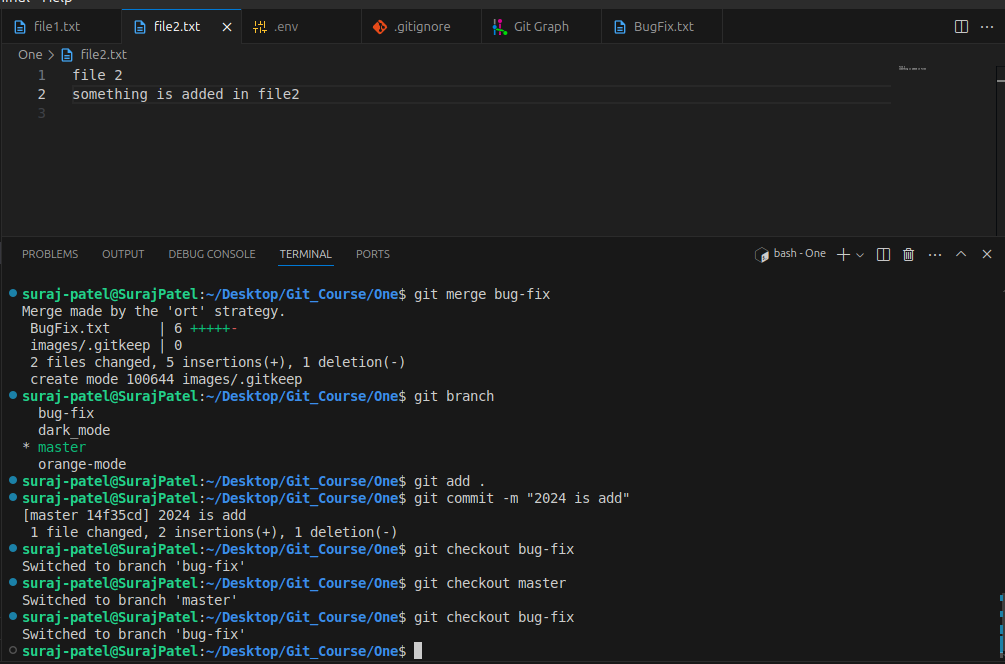
**3.4. Managing conflicts**

There is no magic button to resolve conflicts. You have to manually resolve the conflicts. Decide, what to keep and what to discard. VSCode has a built-in merge tool that can help you resolve the conflicts. I personally use VSCode merge tool. Github also has a merge tool that can help you resolve the conflicts but most of the time I handle them in VSCode and it gives me all the options to resolve the conflicts.

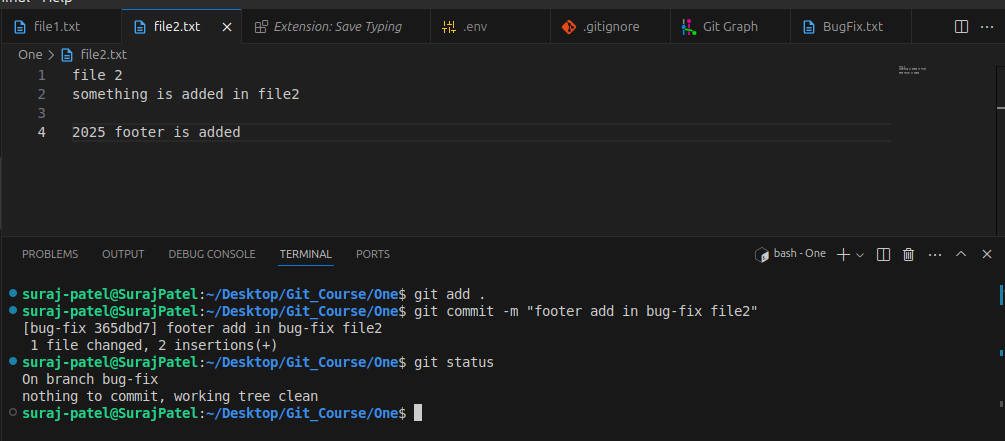
Overall it sounds scary to beginners but it is not, it’s all about communication and understanding the code situation with your team members.

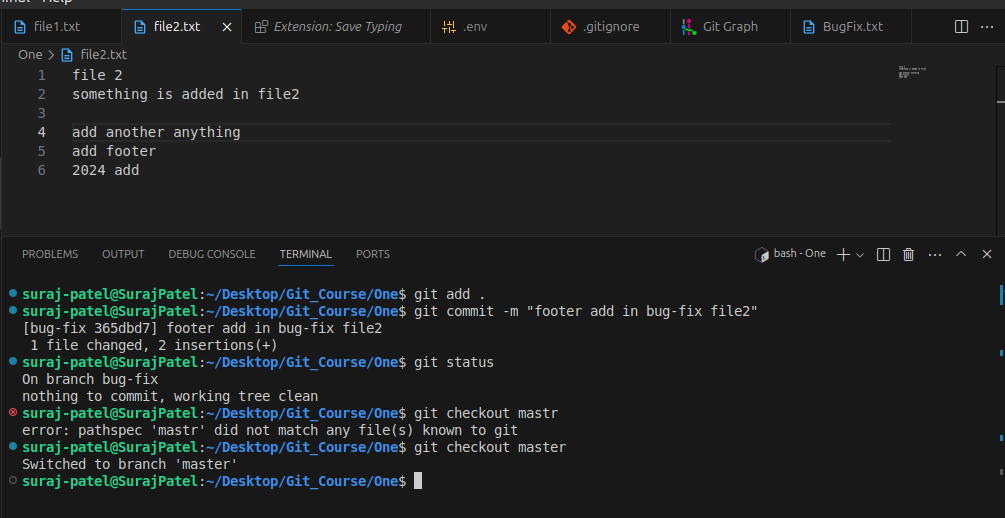
**3.5. Conflicts creation and solving**

In the file2 there is so much content now switching from master(main) to bug-fix branch

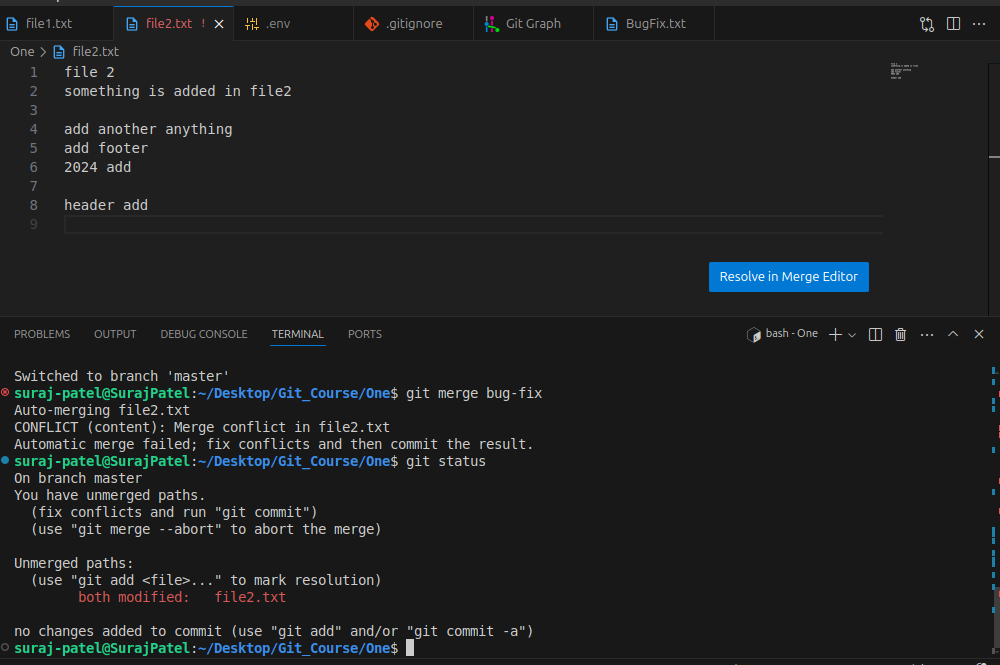


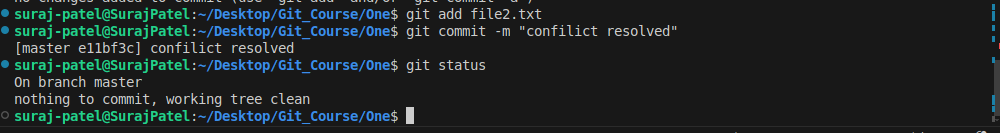
Now we can see that in file2 there is no so much content and now bug-fix added footer also so there is conflict is created

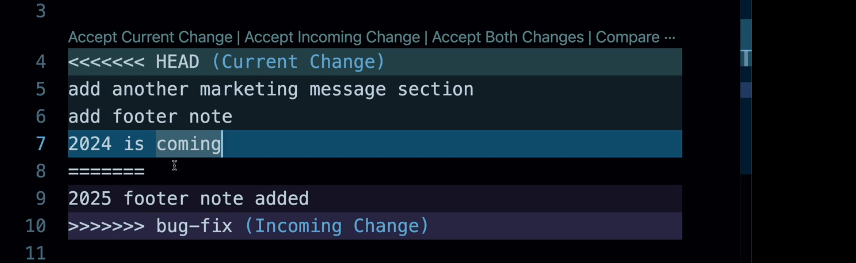
Footer is added not switching to master branch (main branch)

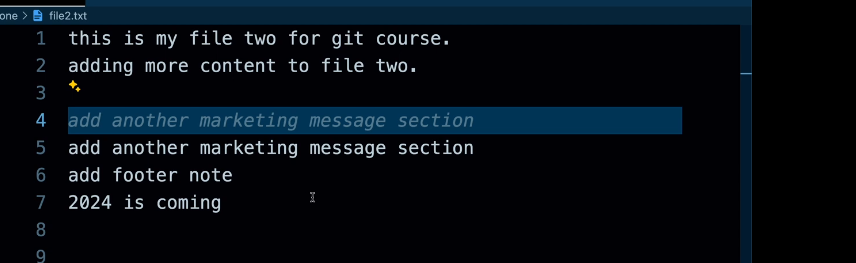
Now you can see there is same file but content is different   
If you try merge with this code git merge bug-fix you get this type of window in vscode

Now, you can select what you want to do from the following options.

now you need to add file and commit

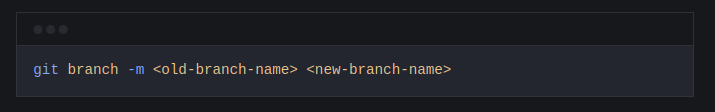
You can also do manually like this

Like this

And now do same thing add and commit

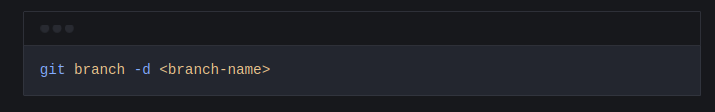
**3.6. Rename a branch**

You can rename a branch using the following command:

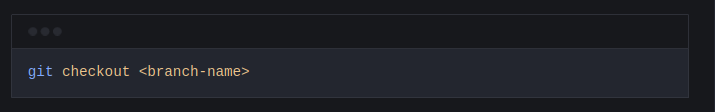


**3.7. Delete a branch**

You can delete a branch using the following command:

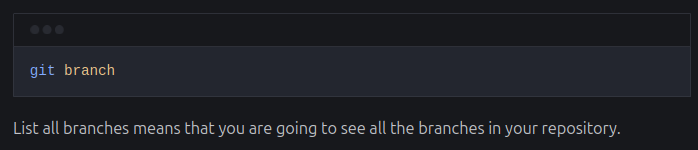
**3.8. Checkout a branch**

You can checkout a branch using the following command:

Checkout a branch means that you are going to work on that branch. You can checkout any branch you want.

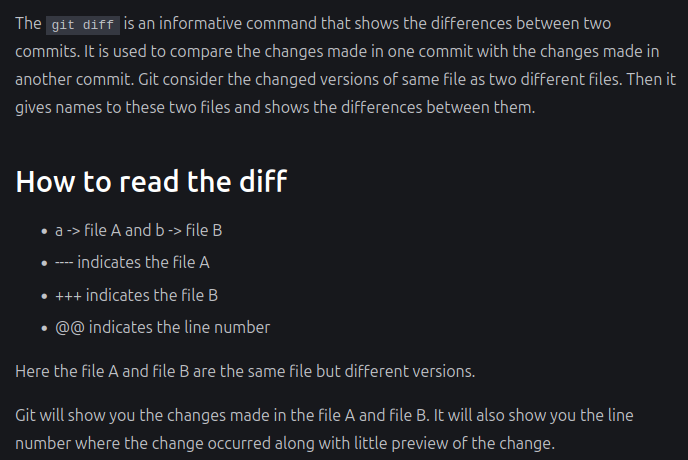
**3.9. List all branches**

You can list all branches using the following command:

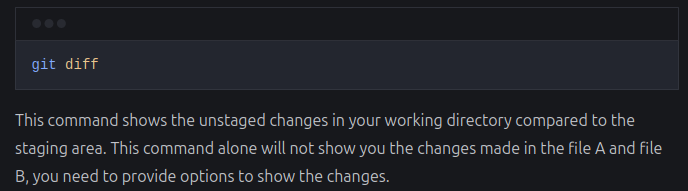


**4. diff stash and Tags**

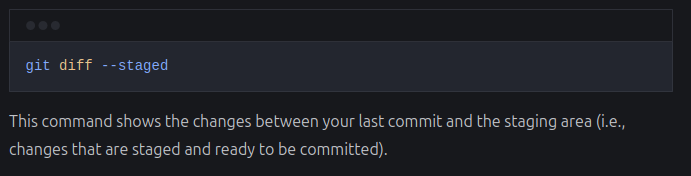
**4.1. Git diff**



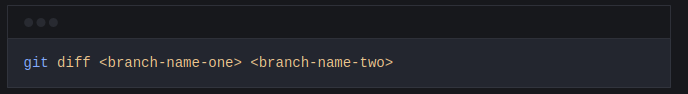
**4.1.1. Comparing Working Directory and Staging Area**



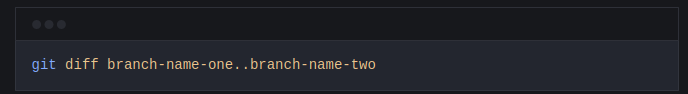
**4.1.2.** **Comparing Staging Area with Repository**



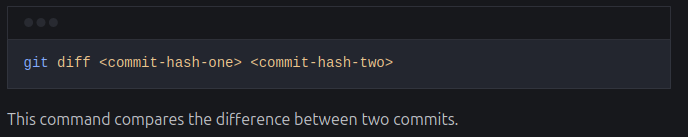
**4.1.3. Comparing between branches**

This command compares the difference between two branches. For example:

Another way to compare the difference between two branches is to use the following command:

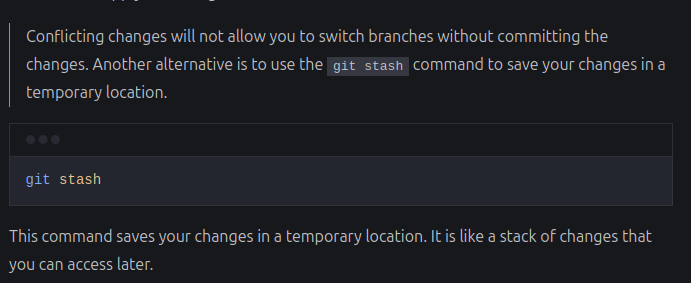


**4.1.4. Comparing Specific Commits:**



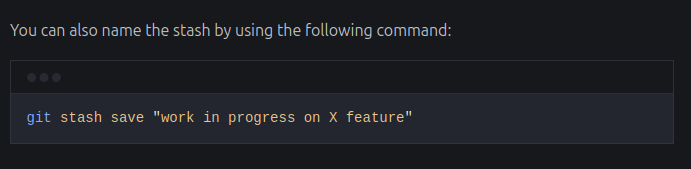
## **4.2. Git Stash**

Stash is a way to save your changes in a temporary location. It is useful when you want to make changes to a file but don’t want to commit them yet. You can then come back to the file later and apply the changes.

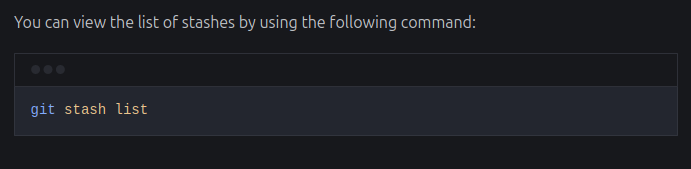
Example:

If you added something in in some file and you want to switch branch then you need to commit first but you want to not commit at that time then we use git stash for temporary save

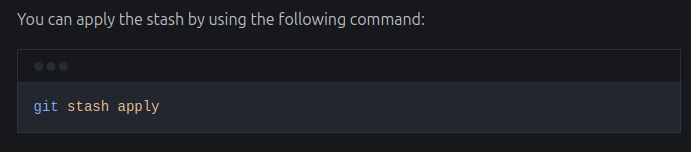
**4.2.1. Naming the stash**



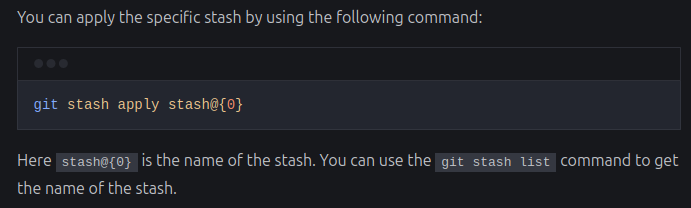
**4.2.2. View the stash list**



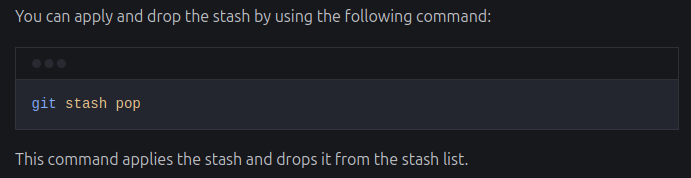
**4.2.3. Apply the stash**



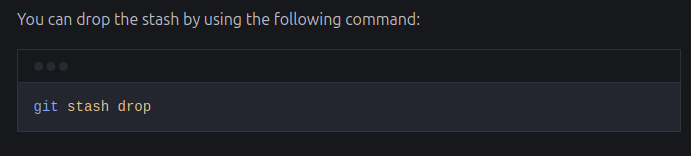
**4.2.4. Apply the specific stash**



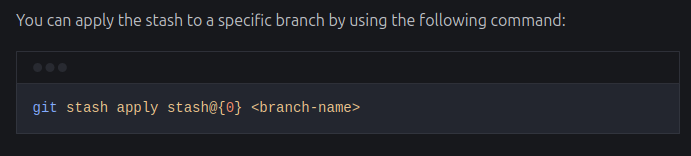
**4.2.5. Applying and dropping the stash**



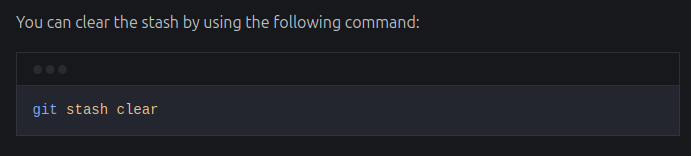
**4.2.6. Drop the stash**



**4.2.7. Applying stash to a specific branch**



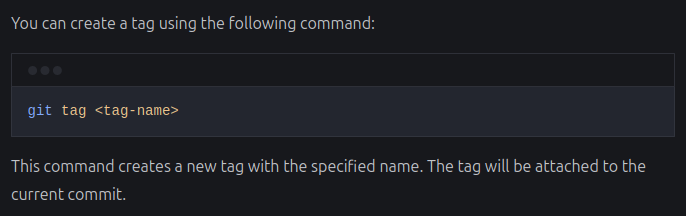
**4.2.8. Clearing the stash**



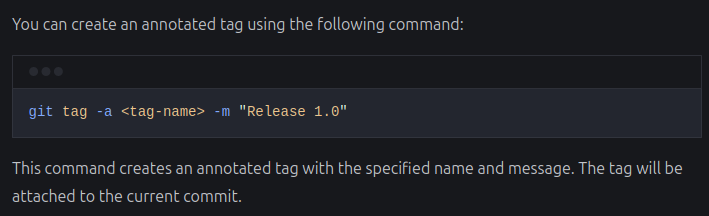
**4.3. Git Tags**

Tags are a way to mark a specific point in your repository. They are useful when you want to remember a specific version of your code or when you want to refer to a specific commit. Tags are like sticky notes that you can attach to your commits.

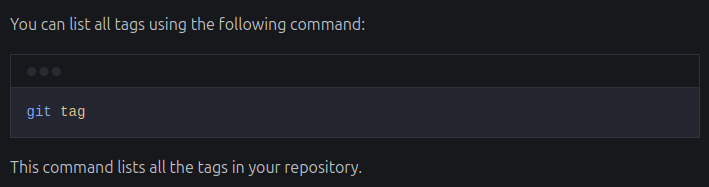
**4.3.1. Creating a tag**



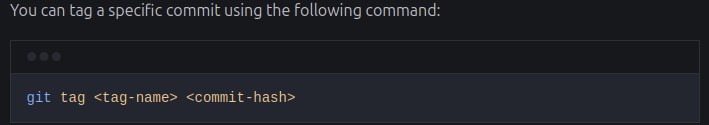
**4.3.2. Create an annotated tag**



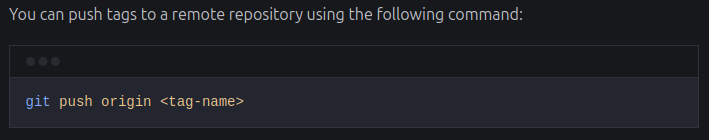
**4.3.3. List all tags**



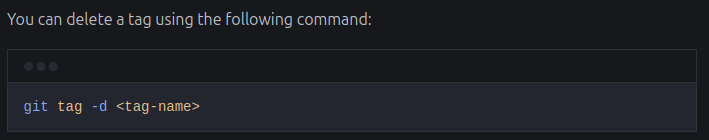
**4.3.4. Tagging a specific commit**



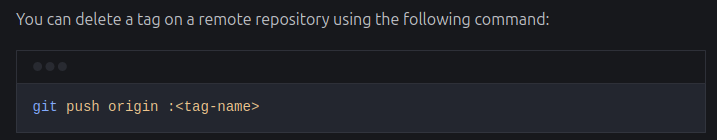
**4.3.5. Push tags to remote repository**



**4.3.6. Delete a tag**



**4.3.7. Delete tag on remote repository**



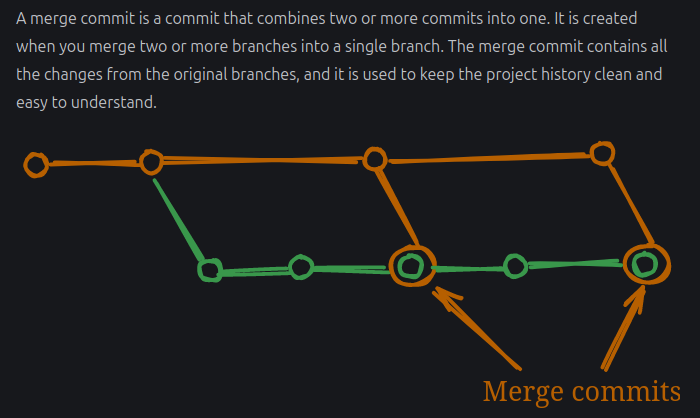
**5. Rebase and reflog**

**5.1. Rebase in git**

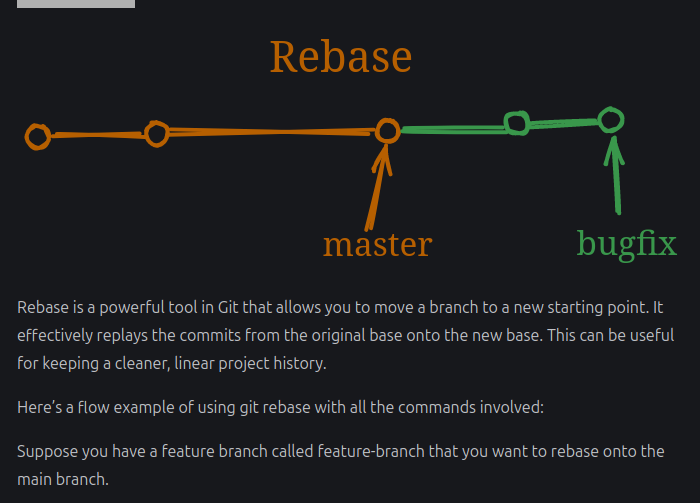
Git rebase is a powerful Git feature used to change the base of a branch. It effectively allows you to move a branch to a new starting point, usually a different commit, by “replaying” the commits from the original base onto the new base. This can be useful for keeping a cleaner, linear project history.

Some people like to use rebase over the merge command because it allows you to keep the commit history cleaner and easier to understand. It also allows you to make changes to the code without affecting the original branch.

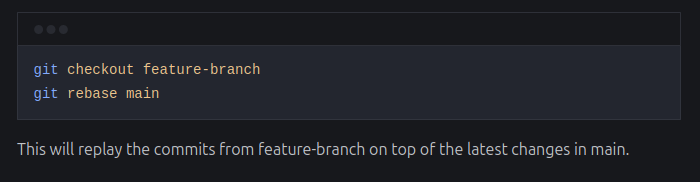
**5.2. Merge commits**



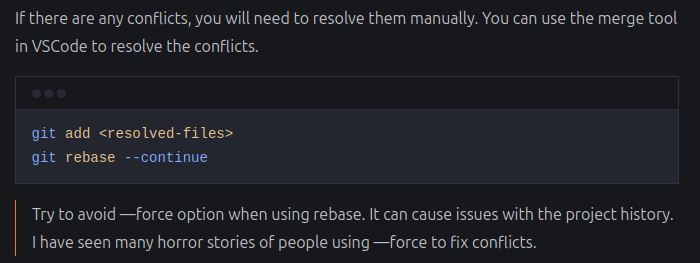
**5.3. Rebase**



**5.4. Ensure you are on the branch you want to rebase:**



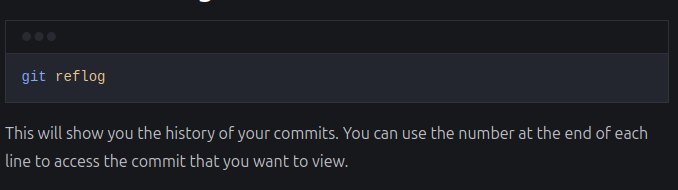
**5.5. Resolve any conflicts:**



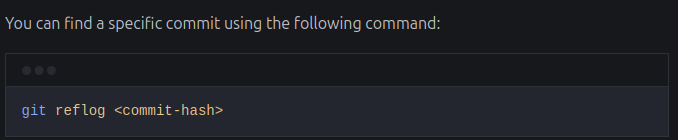
**5.6. Git reflog**

Git reflog is a command that shows you the history of your commits. It allows you to see the changes that you have made to your repository over time. This can be useful for debugging and understanding the history of your project.

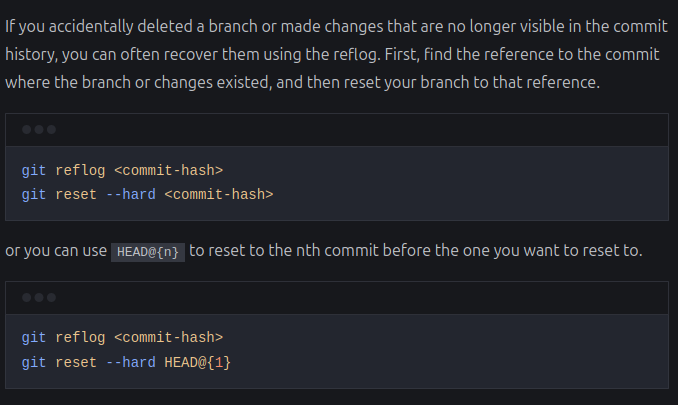
**5.7. View the reflog:**



**5.8. Find specific commit**

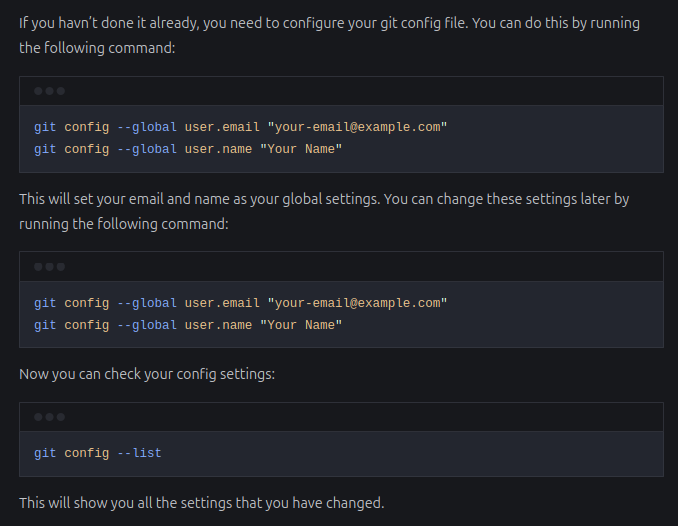


**5.9. Recover lost commits or changes**



**6. Started with Github**

**6.1. Configure your config file**

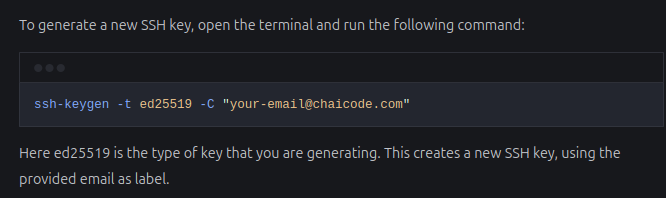


**6.2. Setup ssh key and add to github**

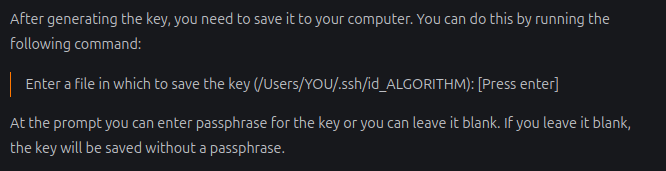
If you haven’t done it already, you need to setup ssh key and add it to your github account. You can do this by following the instructions on the Github website.

You can find the exact steps on the website for both Windows and MacOS. The steps are same for both, only apple users need to add the ssh key to their keychain.

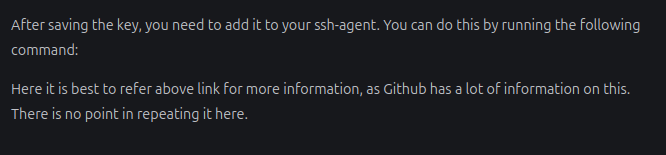
**6.2.1. Step 1: Generate a new SSH key**



**6.2.2. Save the key**



**6.2.3. Add key to your ssh-agent**



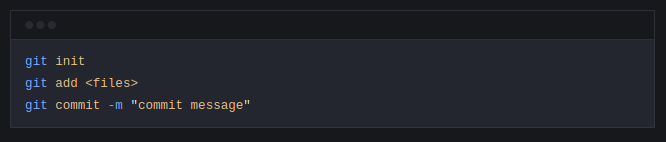
**6.2.4. Add key to github**

Use the webui to add the key to your github account. You can do this by following the instructions on the [Github website](https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account?tool=webui).

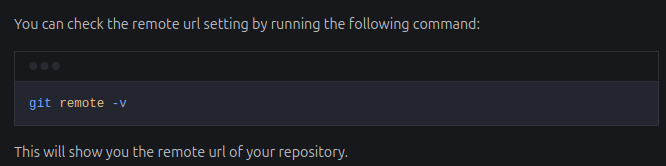
**6.3. Adding code to remote repository**

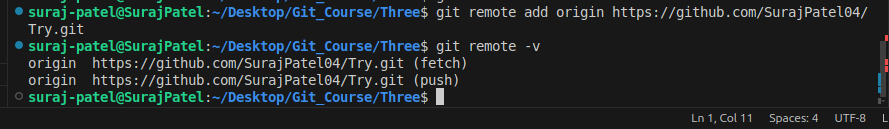
Now that you have setup your ssh key and added it to your github account, you can start pushing your code to the remote repository.

Create a new Repo on your system first, add some code and commit it.

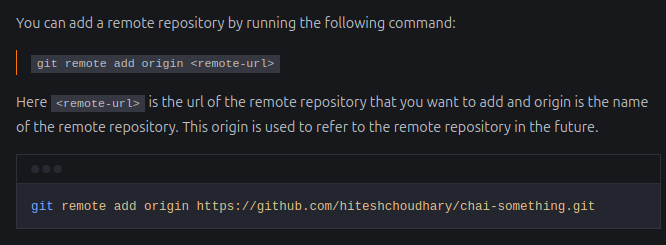


**6.3.1. Check remote url setting**

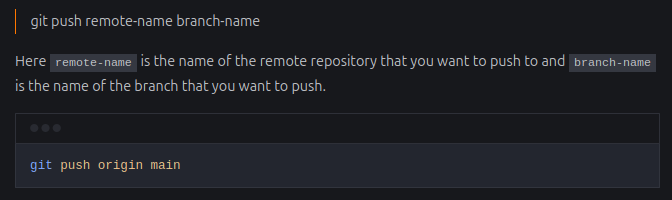
Example after adding remote repository checking and we get this



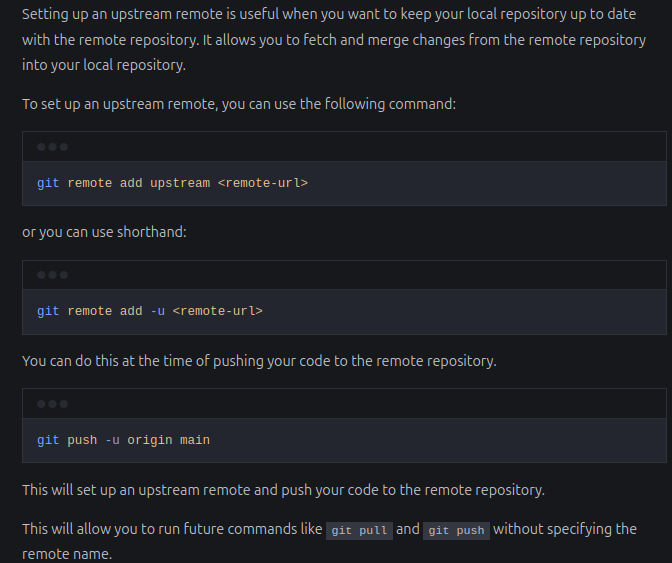
**6.3.2. Add remote repository**



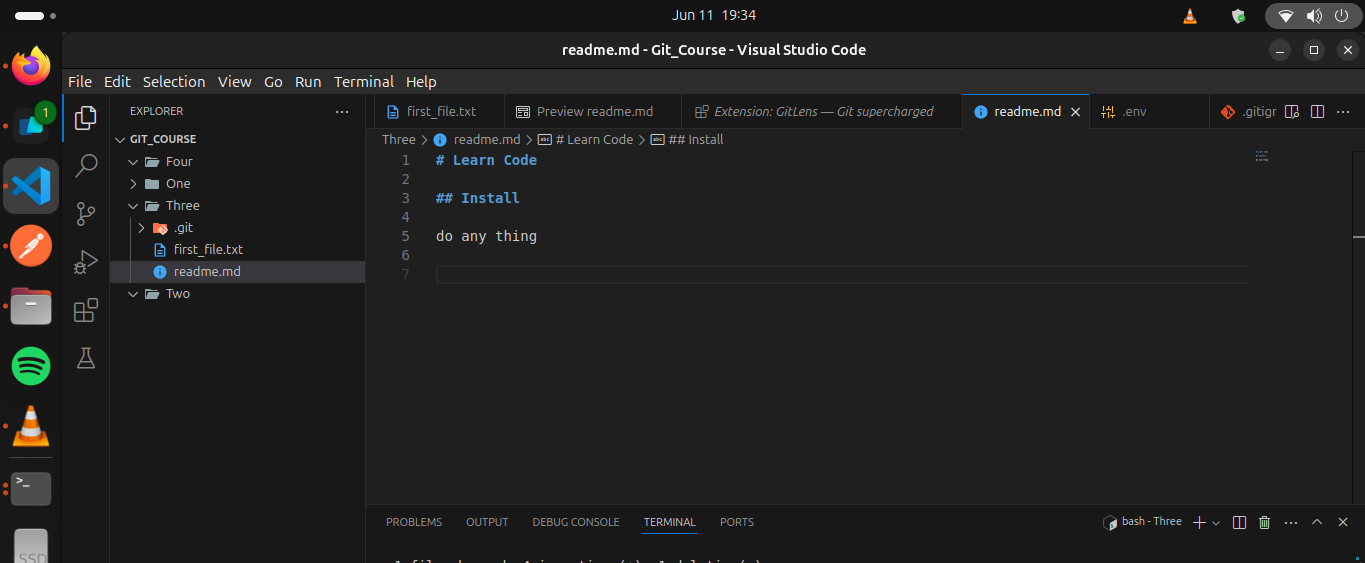
**6.3.3. Push code to remote repository**

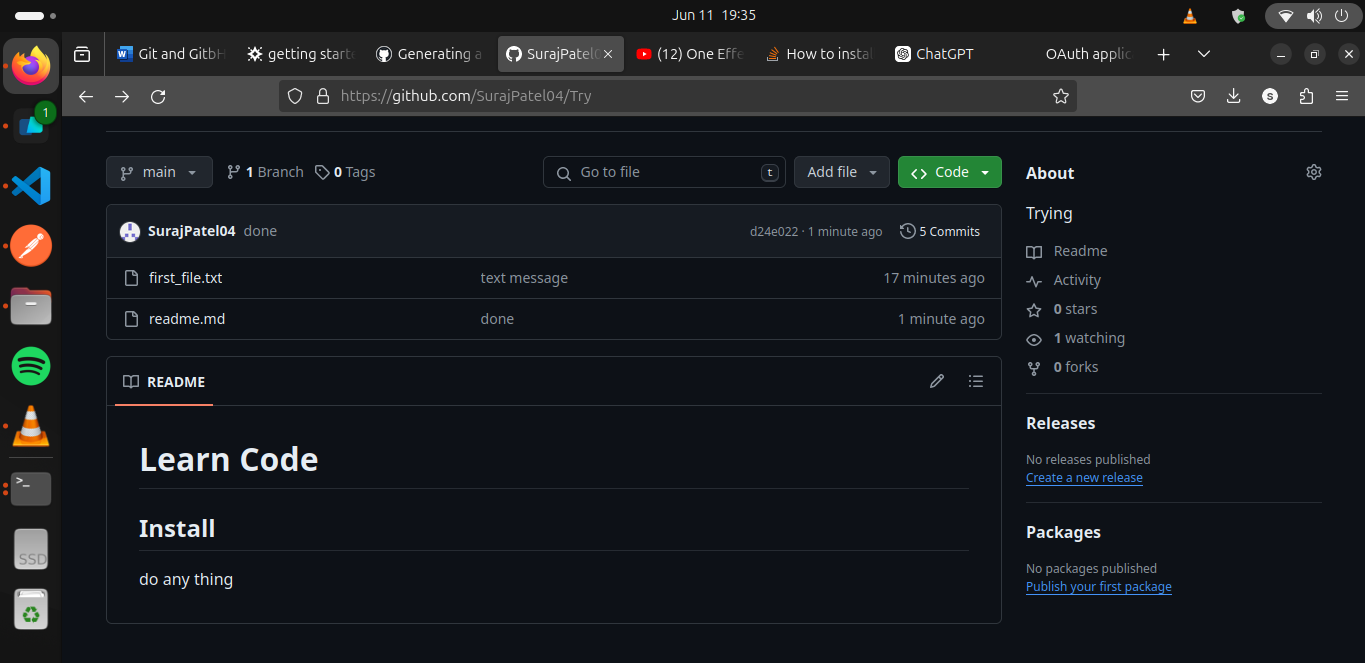
Or use git push –u origin main

**6.3.4. Setup an upstream remote**

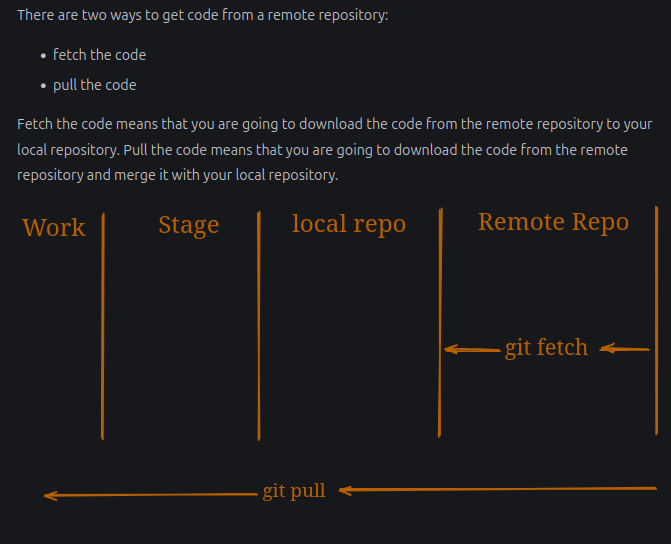


**6.4.** **Readme.md file**

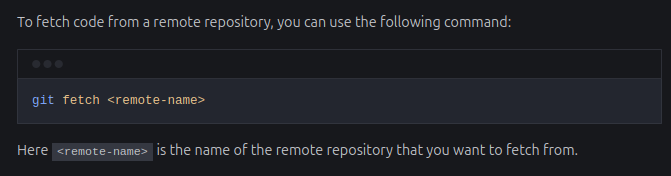
After push you get this



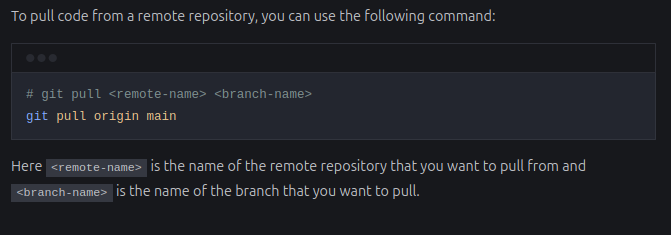
**6.5. Get code from remote repository**



**6.5.1. Fetch code**

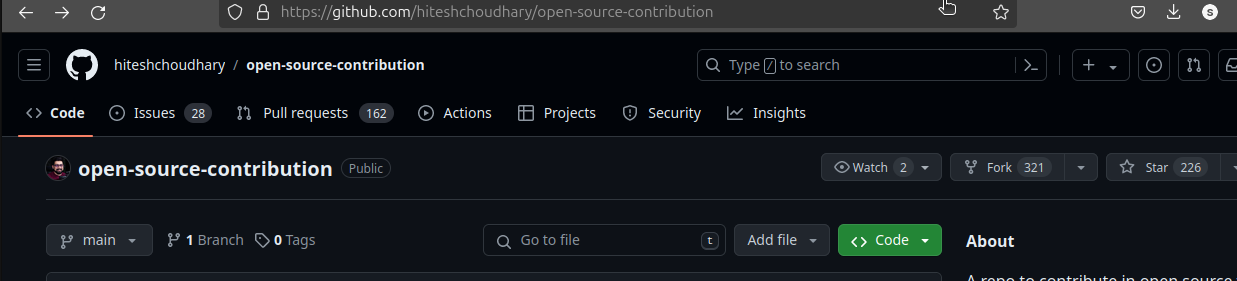


**6.5.2. Pull code**



**7. Open Source**

First step to create Fork

The clone by this step

Copy HTTPS  
and go to vs code and type gti clone <url>

Then to you work and then add commit and push

If you want to send the changes then click on contribution button

