The Complete Guide Git and GitHub

Course Overview:

1. **Introduction to Git and GitHub**
2. **Optional Mac Terminal & Windows Command prompt Introduction**
3. **Version Management with Git – The Basics**
4. **Diving Deeper into the Git**
5. **From Local to Remote Understanding Git Hub**
6. **Git Hub Deep Drive Collaboration & Contribution**
7. **Real Project Example Git & GitHub Applied**

Introduction to Git and GitHub:

What is Git?

Official site: https://git-scm.com/

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Control & tracking of code changes over time

What is Version Management / Control?

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

Fig. Version Management / Control

Fig. Git

Git is a local tool. It's installed on your machine and therefore, only you can use it as the user of this computer. Besides that, the code you have managed in Git can only be accessed from that computer and not from anywhere else. This means if you're not on your computer, and you want to work on a project but you are where, somewhere else in the world, then you have a bit of a problem. Besides the fact that if you lose your computer or if it crashes that the code is lost. But that's another topic. So, Git is maybe just one part of the solution we need to manage our development projects as efficient as possible. This is where GitHub comes into play

What is GitHub?

Official Site: https://github.com/

GitHub is obviously used where the world builds software. We see that millions of developers and companies build, ship, and maintain their software on GitHub.
So, it's the largest and most advanced development platform in the world.
So, GitHub seems to have a high focus on two developers, but what does this mean in detail now? We learnt that GitHub is the largest development platform and that many developers are using it for their projects. And why are developers doing this? Well, because GitHub is a cloud hosting and collaboration provider specifically made for developers, therefore also for web developers. It's Cloud Hosting Provider This means it's a service, which allows you to store your data, not in the local environment, but in the Cloud. It is for free for basic use cases and also the used cases we have here throughout this course. There are paid options mainly required for companies and really bigger companies to well, be able to efficiently manage the project.

GitHub is also for free. And it also is a collaboration provider. This means which Git is a local tool, GitHub with the code being available in the Cloud, also allows us to collaborate on our projects with other people. And this is very important because if you think about big companies like Facebook, for example, or also smaller companies for two, three, four, five or 10 developers work together on the same project, well, these developers or we as developers need access to this code to work on their specific parts of a website, for example. And this is what GitHub does in the end. **GitHub is a Git repository hosting provider** to be even more precise here. A Git repository is basically a Git-managed project in simple terms. So with that capabilities, so these local capabilities of Git with managing that code efficiently, with managing the history and also with tracking changes, and the capabilities of GitHub with being able to host the code on a Cloud service and to enable that great collaboration capabilities, well, this is how Git and GitHub are connected.
It's very important to understand though, that GitHub and Git are not generally related. This relation only evolve because well, GitHub is the perfect addition to Git, and turns out that many people using it are also using GitHub, therefore this is why Git and GitHub sometimes are assumed to be well kind of one thing. But these are two different tools, two different services, perfectly working and perfectly used together by millions of developers in the world. So, this is what Git and GitHub are in a nutshell.

Fig. Git & GitHub.

Optional Mac Terminal & Windows Command prompt Introduction:

The Text Based Computer Interaction (Command Line What & Why?)

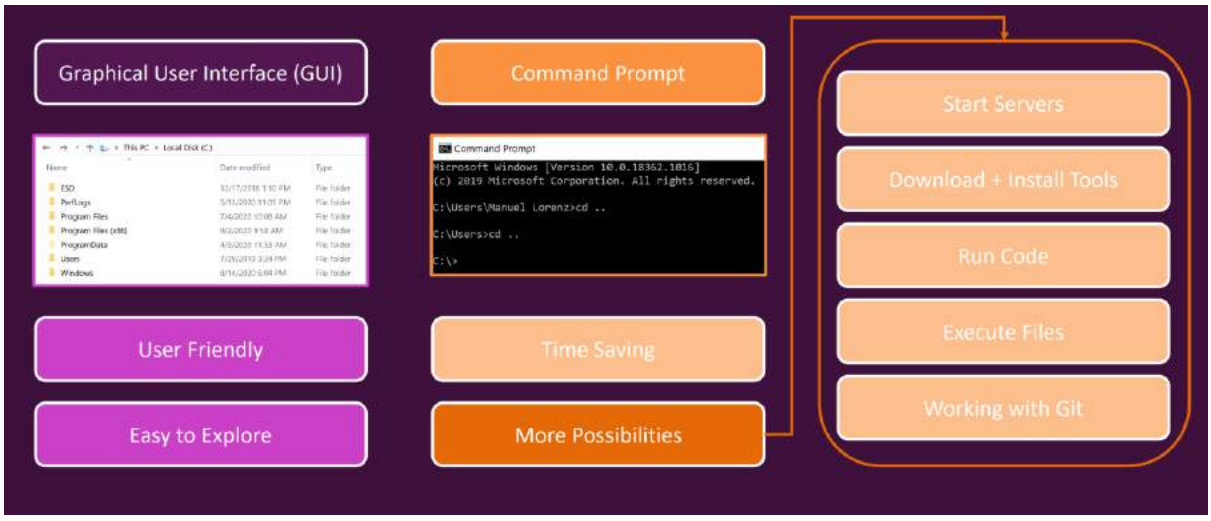


Fig. GUI vs Command Prompt

Comparing Mac & Windows Command Line:

Fig. Mac Terminology

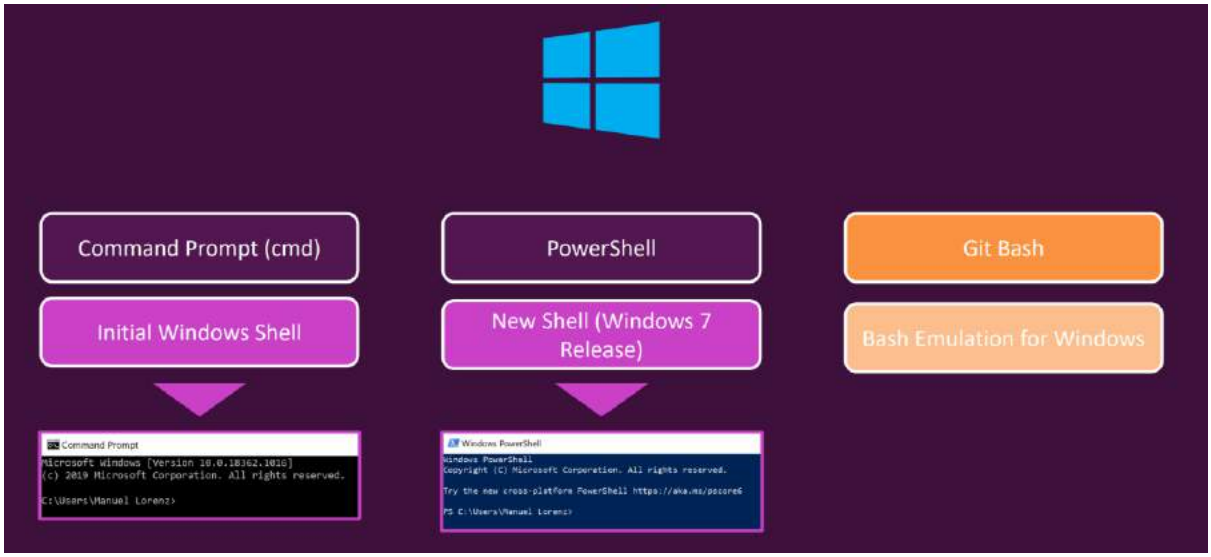


Fig. Windows Terminology

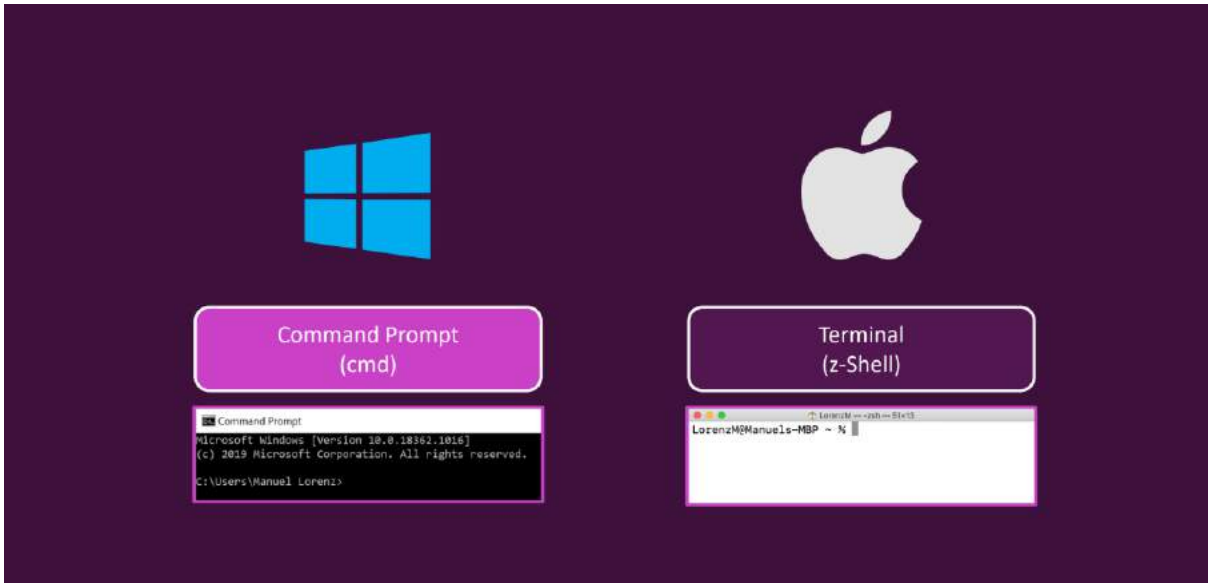


Fig. Command Line Tools

Mac Terminal:

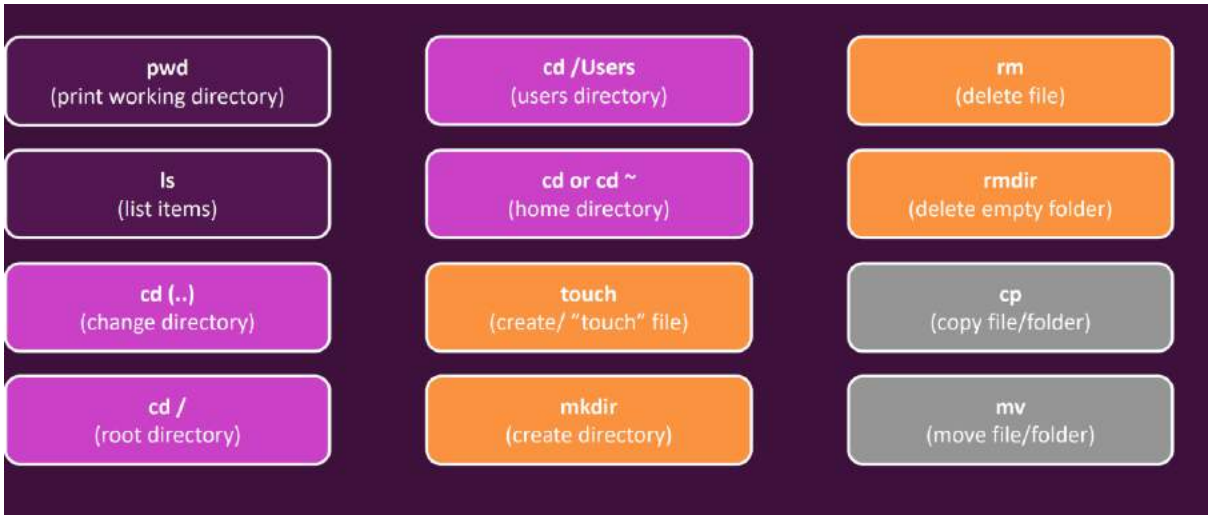


Fig. Mac Terminal (z-Shell Command)

Note: This Command Used in Mac-Book for Mac User.

Windows Command Prompt:

Command Prompt is the traditional Command Line Interface

The Basics:

- **dir: lists items**

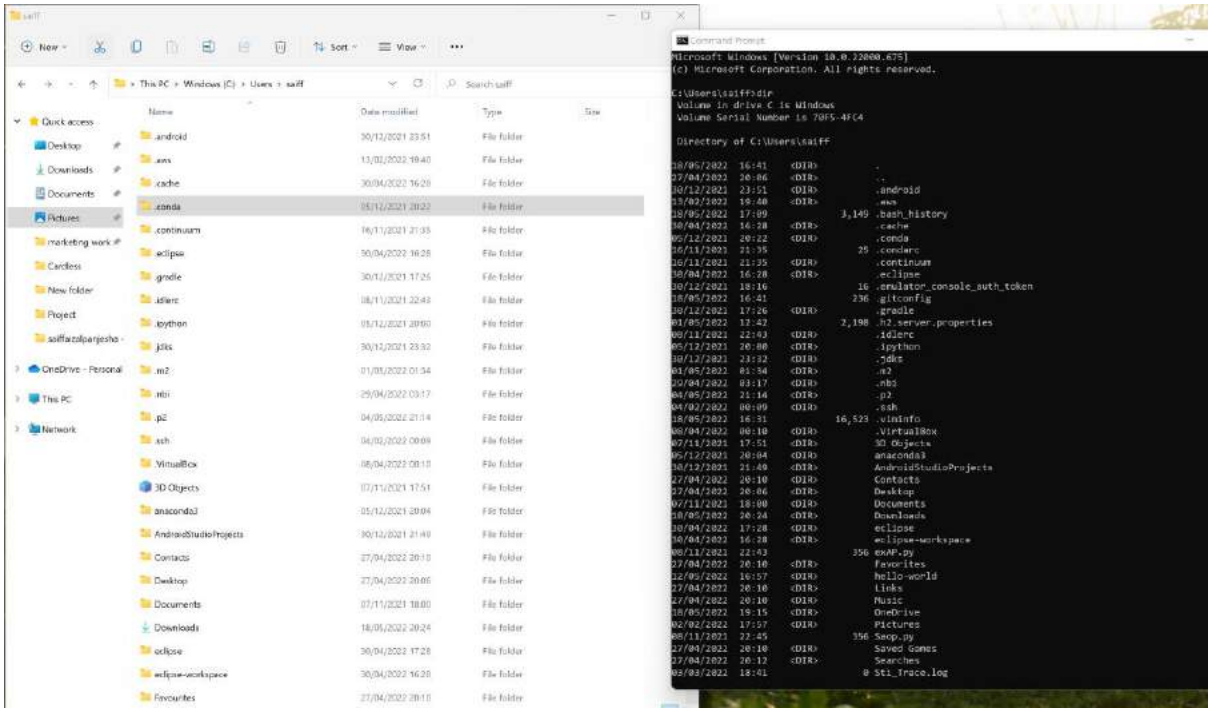


Fig. Lists of Items

- **cd(..): Change Directory**

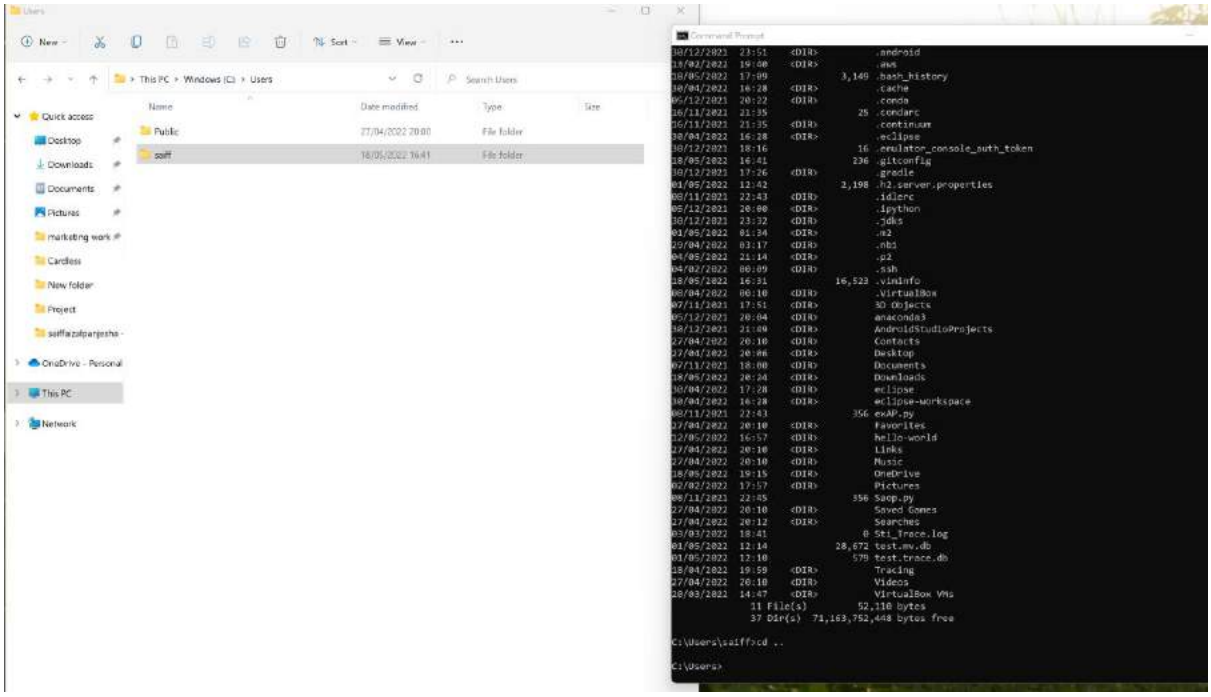


Fig. Change Directory to Users

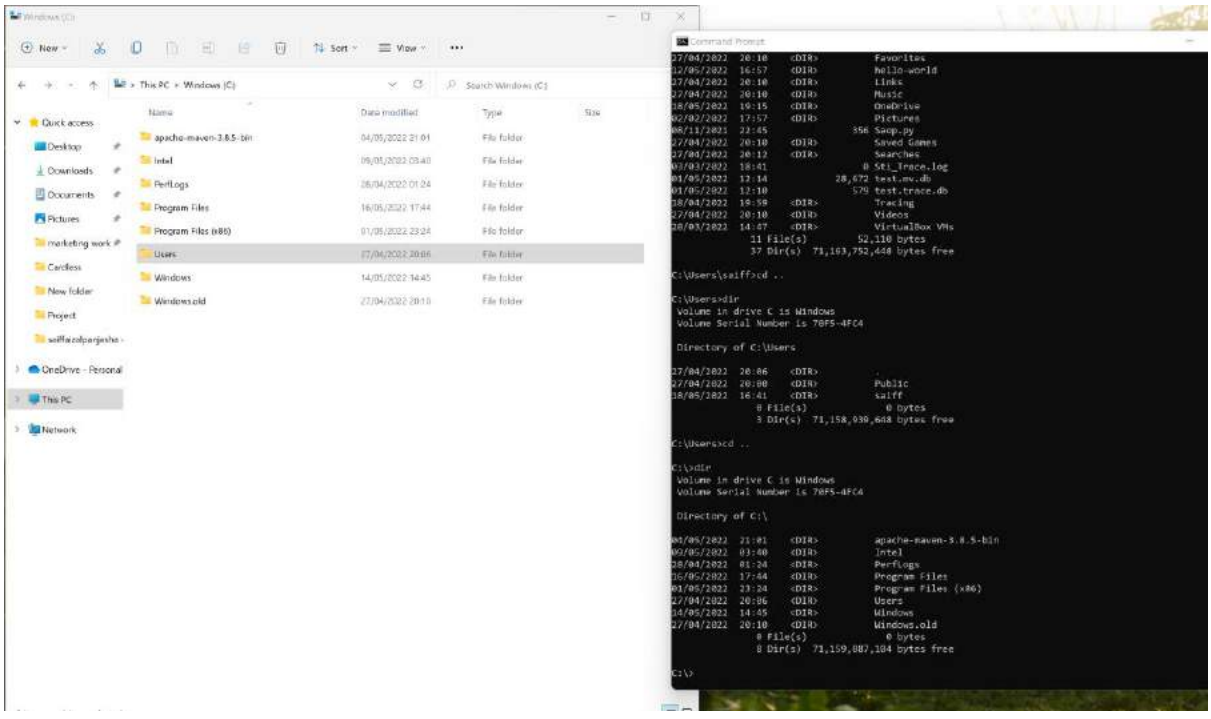


Fig. Change Directory to C drive

Changing to D- Drive:

C:\>D:

D:\>

- **Relative Paths:**

D:\>cd Devops

```
D:\Devops>dir
 Volume in drive D is Data
 Volume Serial Number is 04F5-839B

 Directory of D:\Devops

18/05/2022  18:08    <DIR>          .
02/05/2022  17:35    <DIR>          hello-world
18/05/2022  18:08    <DIR>          Project
18/05/2022  17:49    <DIR>          saiffaizalpanjesha -aws
               0 File(s)              0 bytes
               4 Dir(s)  275,885,756,416 bytes free
```

- **Absolute Path:**

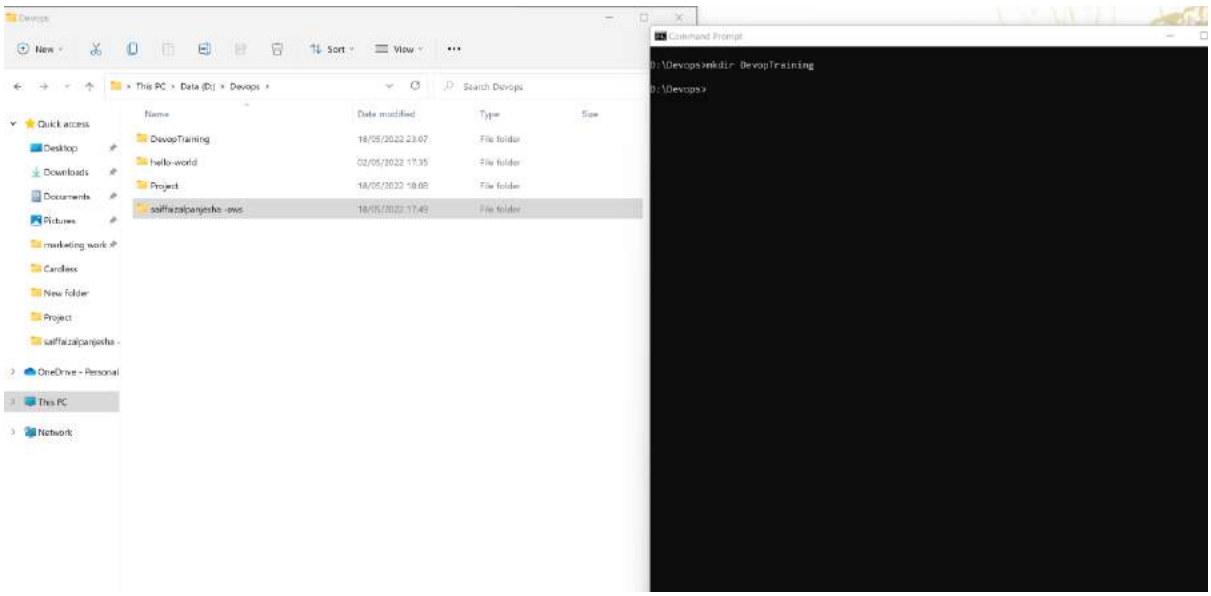
```
D:\Devops>cd D:\Devops\saiffaizalpanjesha -aws

D:\Devops\saiffaizalpanjesha -aws>dir
 Volume in drive D is Data
 Volume Serial Number is 04F5-839B
 Directory of D:\Devops\saiffaizalpanjesha -aws
18/05/2022  17:49    <DIR>          .
18/05/2022  18:08    <DIR>          ..
12/04/2022  08:49            26,740 3646_Skills.pdf
18/05/2022  17:49               252 Devops Other Courses.txt
02/05/2022  00:01             1,700 Devops_Project_Key.pem
18/05/2022  02:25               176 devpos.txt
01/05/2022  23:47               135 new_user_credentials.csv
03/05/2022  21:57             1,868 Password.txt
               6 File(s)         30,871 bytes
               2 Dir(s)  275,885,756,416 bytes free
```

- **cls :(Clear command prompt)**

Fig. Clear Screen of Cmd Prompt

- **mkdir : Creates Folder**
- **echo our Devops File > text.txt : Creating a File**



```
D:\Devops\DevopTraining>echo our Devops File > text.txt

D:\Devops\DevopTraining>dir
 Volume in drive D is Data
 Volume Serial Number is 04F5-839B

 Directory of D:\Devops\DevopTraining

18/05/2022  23:11    <DIR>          .
18/05/2022  23:07    <DIR>          ..
18/05/2022  23:11                18 text.txt
               1 File(s)             18 bytes
               2 Dir(s)  275,885,752,320 bytes free

D:\Devops\DevopTraining>type text.txt
our Devops File

D:\Devops\DevopTraining>
```

Fig. Create a Folder and File (Devop Training and text)

- **del : Deleting File**

```
D:\Devops\DevopTraining>del text.txt

D:\Devops\DevopTraining>dir
 Volume in drive D is Data
 Volume Serial Number is 04F5-839B

 Directory of D:\Devops\DevopTraining

18/05/2022  23:15    <DIR>          .
18/05/2022  23:07    <DIR>          ..
               0 File(s)              0 bytes
               2 Dir(s)  275,885,539,328 bytes free

D:\Devops\DevopTraining>
```

Fig. Deleted File Text

- **rmdir: Deleting Folder**

```
D:\Devops\DevopTraining>cd ..

D:\Devops>rmdir DevopTraining

D:\Devops>dir
 Volume in drive D is Data
 Volume Serial Number is 04F5-839B

 Directory of D:\Devops

18/05/2022  23:18    <DIR>          .
02/05/2022  17:35    <DIR>          hello-world
18/05/2022  18:08    <DIR>          Project
18/05/2022  17:49    <DIR>          saiffaizalpanjesha -aws
               0 File(s)              0 bytes
               4 Dir(s)  275,885,506,560 bytes free

D:\Devops>
```

Fig. Deleted Folder Devop Training

- **copy : Copying Files**

```
D:\Devops>mkdir others

D:\Devops>mkdir main

D:\Devops>echo Hello Miss > test2.txt

D:\Devops>dir
 Volume in drive D is Data
 Volume Serial Number is 04F5-8398

 Directory of D:\Devops

18/05/2022  23:23    <DIR>          .
02/05/2022  17:35    <DIR>          hello-world
18/05/2022  23:23    <DIR>          main
18/05/2022  23:23    <DIR>          others
18/05/2022  18:08    <DIR>          Project
18/05/2022  17:49    <DIR>          saiffaizalpanjesha -aws
18/05/2022  23:23                13 test2.txt
               1 File(s)             13 bytes
               6 Dir(s)  275,885,506,560 bytes free

D:\Devops>type test2
The system cannot find the file specified.

D:\Devops>type test2.txt
Hello Miss

D:\Devops>copy test2.txt main
        1 file(s) copied.

D:\Devops>
```

Fig. Copying Files

- **move: Move the Files**

```
D:\Devops\main>del test2.txt

D:\Devops\main>dir
 Volume in drive D is Data
 Volume Serial Number is 04F5-8398

 Directory of D:\Devops\main

18/05/2022  23:30    <DIR>          .
18/05/2022  23:23    <DIR>          ..
               0 File(s)              0 bytes
               2 Dir(s)  275,885,371,392 bytes free

D:\Devops\main>cd ..

D:\Devops>move test2.txt main
        1 file(s) moved.

D:\Devops>dir
 Volume in drive D is Data
 Volume Serial Number is 04F5-8398

 Directory of D:\Devops

18/05/2022  23:31    <DIR>          .
02/05/2022  17:35    <DIR>          hello-world
18/05/2022  23:31    <DIR>          main
18/05/2022  23:23    <DIR>          others
18/05/2022  18:08    <DIR>          Project
18/05/2022  17:49    <DIR>          saiffaizalpanjesha -aws
               0 File(s)              0 bytes
               6 Dir(s)  275,885,371,392 bytes free

D:\Devops>cd main

D:\Devops\main>dir
 Volume in drive D is Data
 Volume Serial Number is 04F5-8398

 Directory of D:\Devops\main

18/05/2022  23:31    <DIR>          .
18/05/2022  23:31    <DIR>          ..
18/05/2022  23:23                13 test2.txt
               1 File(s)             13 bytes
               2 Dir(s)  275,885,371,392 bytes free

D:\Devops\main>type test2.txt
Hello Miss
```

Fig. Moving Files

Version Management with Git – The Basics

Contents:

- **Theory – How Git Works?**
- **Installation & Deployment Environments**
- **Repositories, Branches and Commit**

How Git Works?

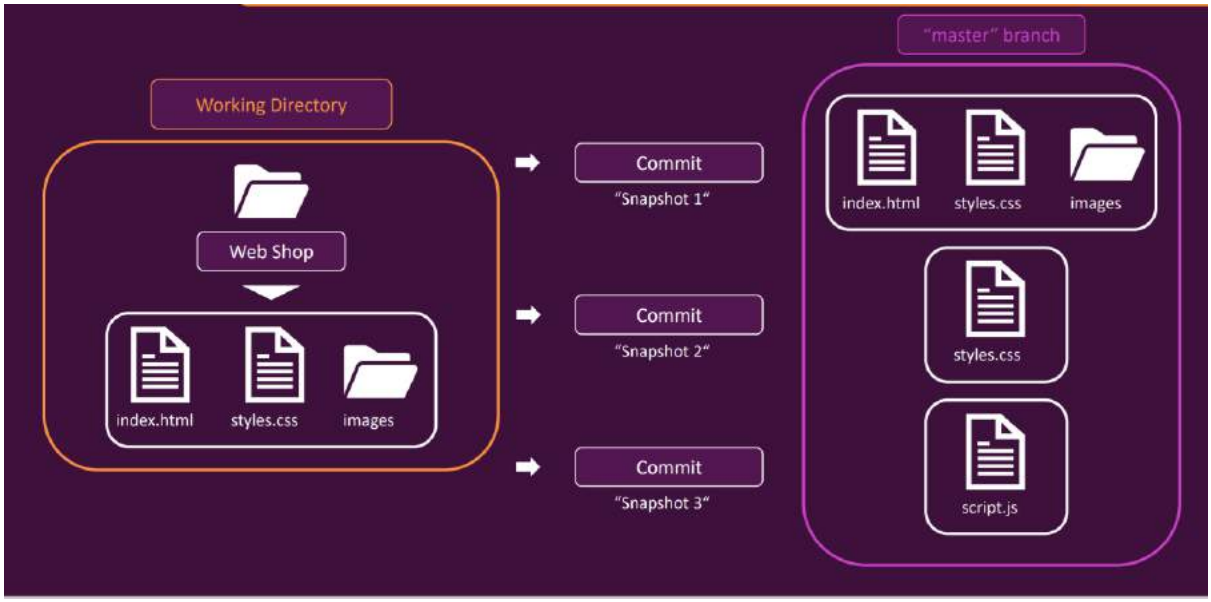


Fig. Working of Git.

Working Directory vs Repository

Git under the Hood:

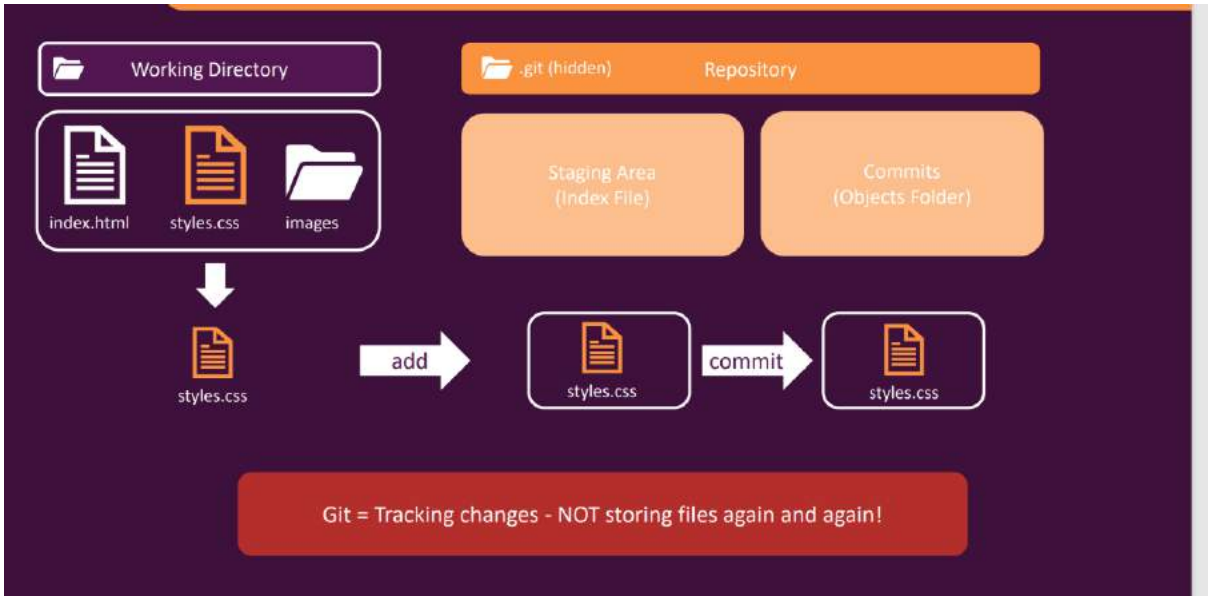


Fig. Git Under the Hood (Directory & Repositories)

Understanding Branches & Commit

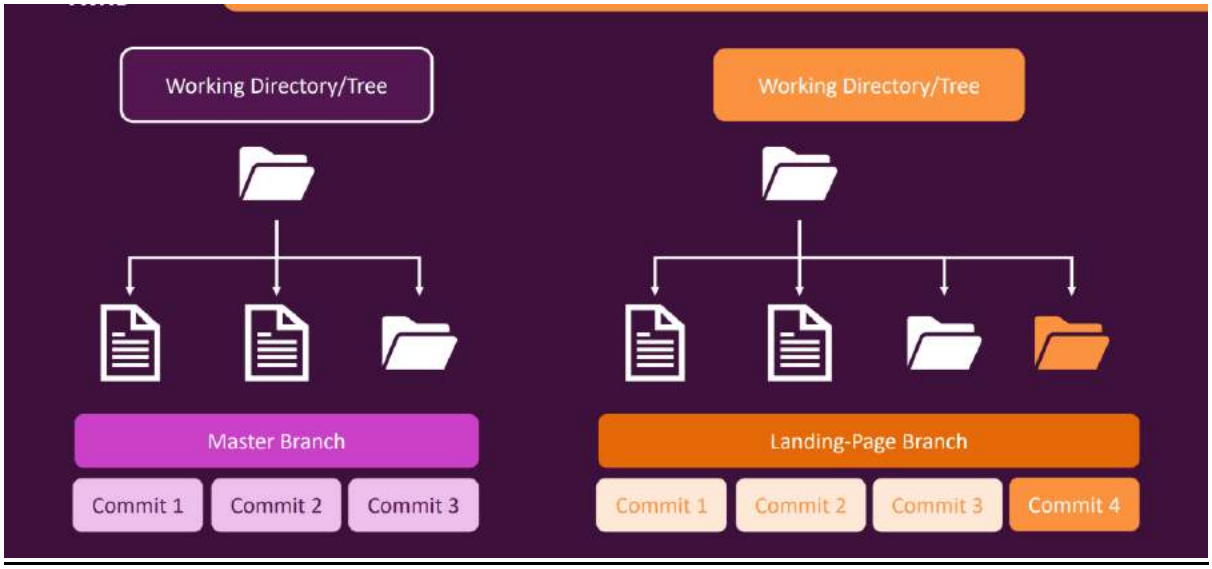


Fig. Branches & Commit

Installing Git on Windows:

Go to official website: https://git-scm.com/

1. Steps For Installing Git for Windows. Download Git for Windows. Extract and Launch Git Installer. Server Certificates, Line Endings and Terminal Emulators. ...

2. How to Launch Git in Windows. Launch Git Bash Shell. Launch Git GUI.

3. Connecting to a Remote Repository. Create a Test Directory. Configure GitHub Credentials.

Fig. Successful Installed Git version 2.35.1

Installing Visual Studio Code:

Go to Website: https://code.visualstudio.com/

Step 1: Download VS code from here Link.

Step 2: Download the Visual Studio Code installer for Windows. Once it is downloaded, run the installer (VSCodeUserSetup-{version}.exe). Then, run the file – it will only take a minute. Accept the agreement and click “next.”

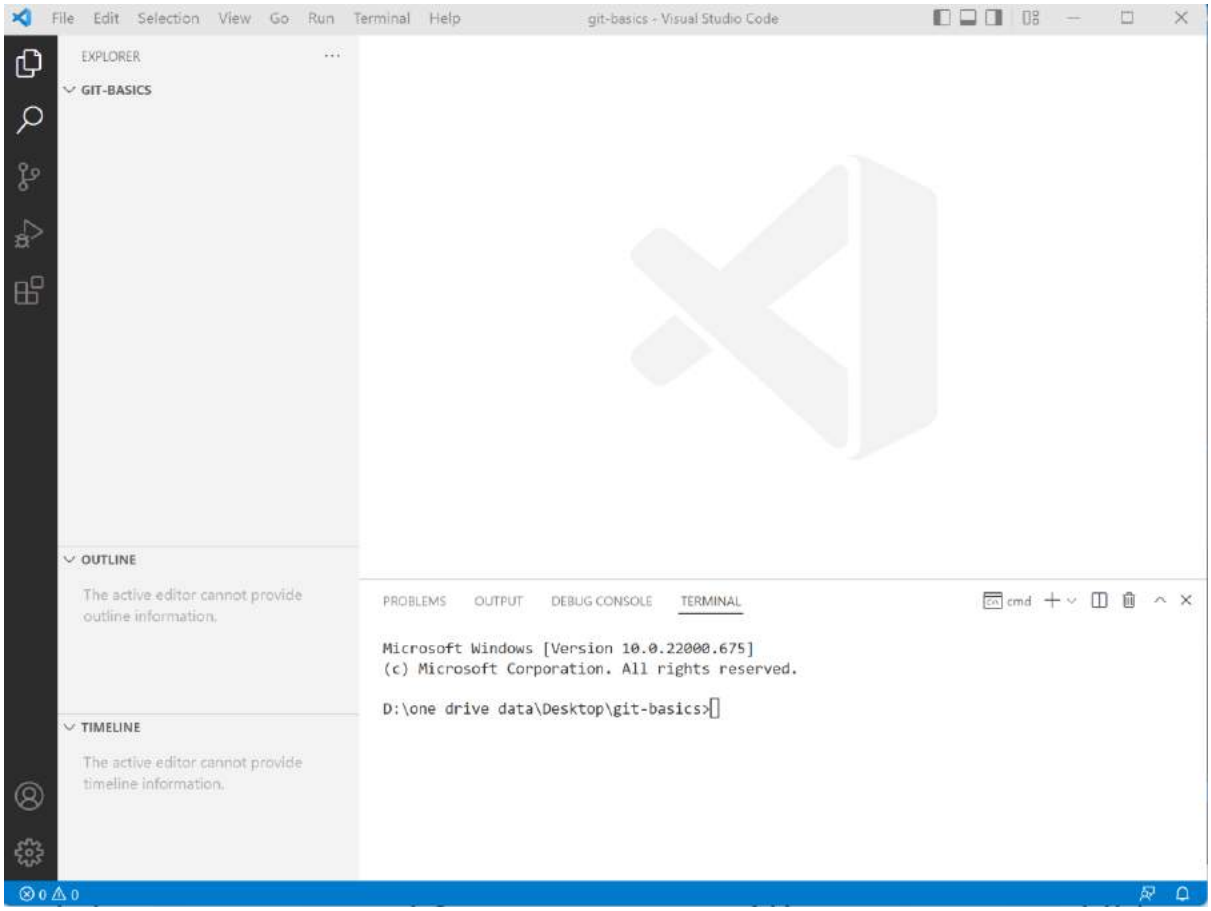


Fig. Visual Studio Install Success

Initializing the repository & creating the First Commit ("git init" and "git commit"):

D:\one drive data\Desktop\git-basics>git --version

git version 2.35.1.windows.2

D:\one drive data\Desktop\git-basics>git status

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

D:\one drive data\Desktop\git-basics>

git init : Initialize the Git

D:\one drive data\Desktop\git-basics>git init

Initialized empty Git repository in D:/one drive data/Desktop/git-basics/.git/

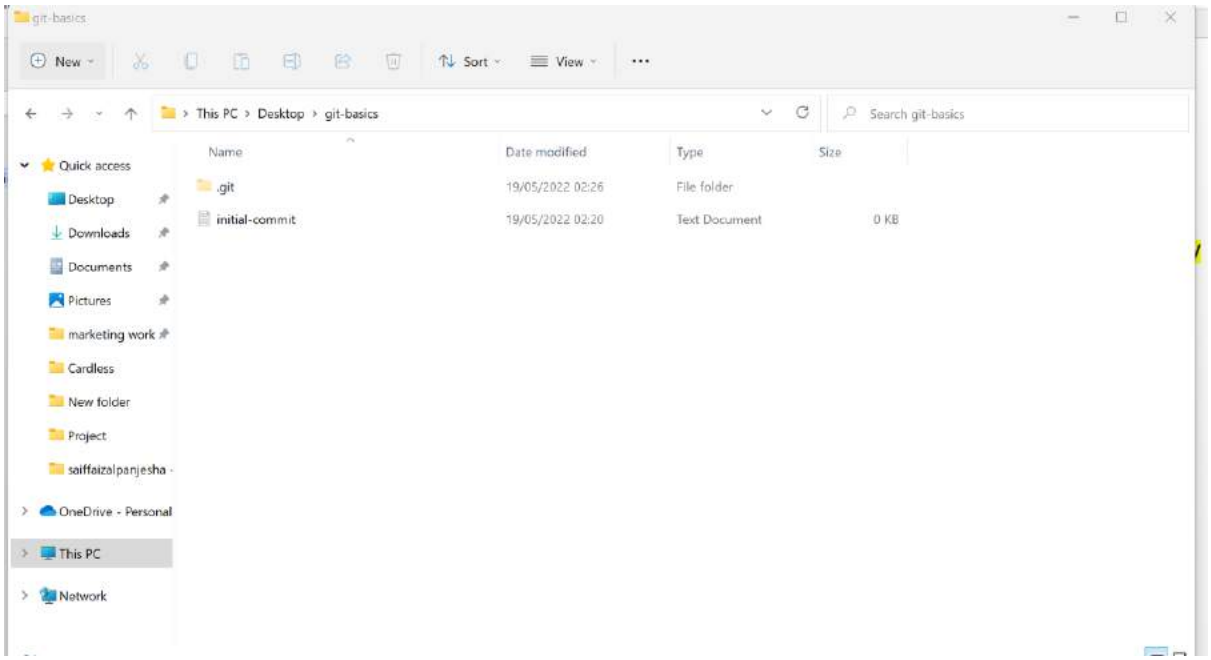


Fig. git hidden folder created after initialized

Untracked Files :

D:\one drive data\Desktop\git-basics>git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

initial-commit.txt

nothing added to commit but untracked files present (use "git add" to track)

Tracking the File:

D:\one drive data\Desktop\git-basics>git add . //Adding the files

D:\one drive data\Desktop\git-basics>git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: initial-commit.txt // initiated after added

Configure your Git username/email

1. Open the command line.
2. Set your username: git config --global user.name "FIRST_NAME LAST_NAME"
3. Set your email address: git config --global user.email "MY_NAME@example.com"

D:\one drive data\Desktop\git-basics>git commit -m "updated file.txt"

[master (root-commit) 3207266] updated file.txt

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 initial-commit.txt

D:\one drive data\Desktop\git-basics>git status

On branch master

nothing to commit, working tree clean

Diving Deeper into Commit with “git log”

To Check where is Commit History?

D:\one drive data\Desktop\git-basics>git log

commit 320726669068faa962f8e245df7e3be52c76accc (HEAD -> master)

Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>

Date: Thu May 19 02:36:24 2022 +0530

updated file.txt

Create a second Commit File:

Fig. Second Commit in git

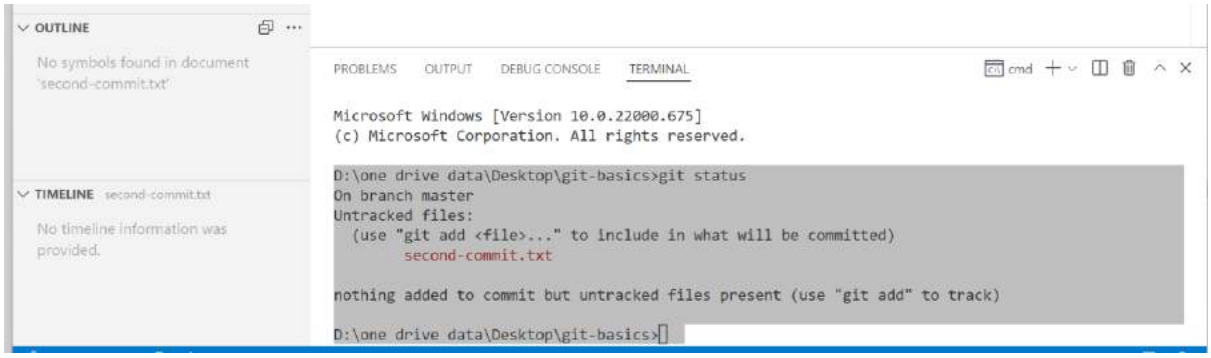


Fig. git status for untracked file

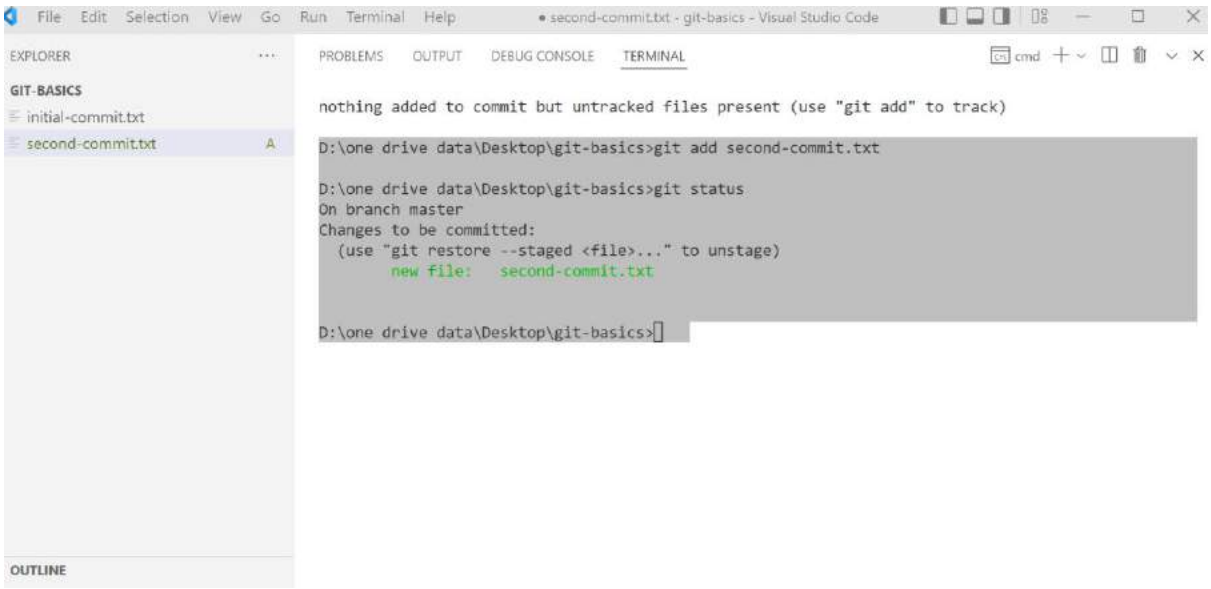


Fig. "git add" to track file

The git commit command captures a snapshot of the project's currently staged changes.

```
D:\one drive data\Desktop\git-basics>git commit -m "added second-commit.txt"
[master dc4a671] added second-commit.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 second-commit.txt

D:\one drive data\Desktop\git-basics>
```

Fig. Staged Changes “git -commit”

```
 create mode 100644 second-commit.txt

D:\one drive data\Desktop\git-basics>git log
commit dc4a67141cb3c3529cdc7da6200e9600f0c56387 (HEAD -> master)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Thu May 19 15:02:23 2022 +0530

    added second-commit.txt

commit 320726669068faa962f8e245df7e3be52c76accc
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Thu May 19 02:36:24 2022 +0530

    updated file.txt
```

Fig. “git log “jump back to history

To check Back the previous commit

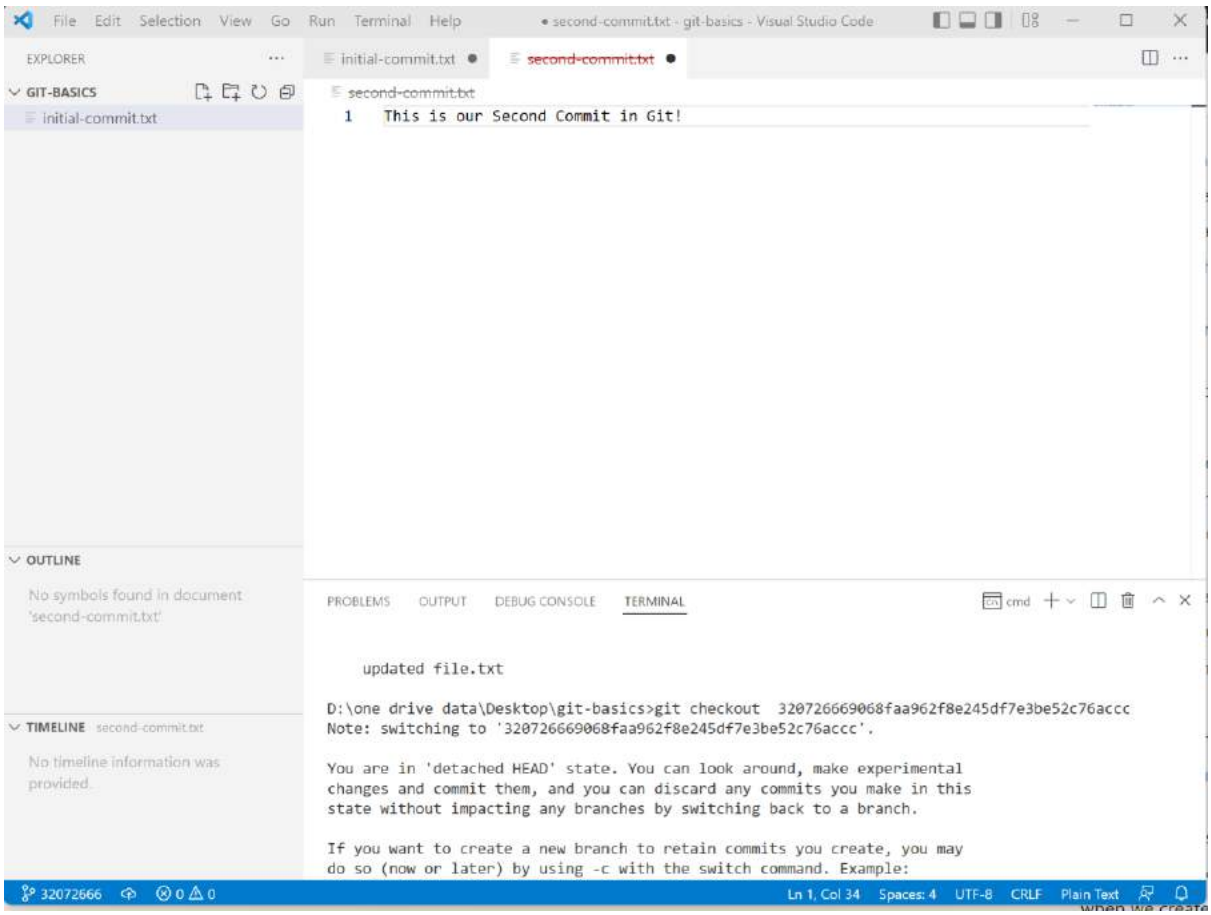


Fig. “Checkout our initial commit”

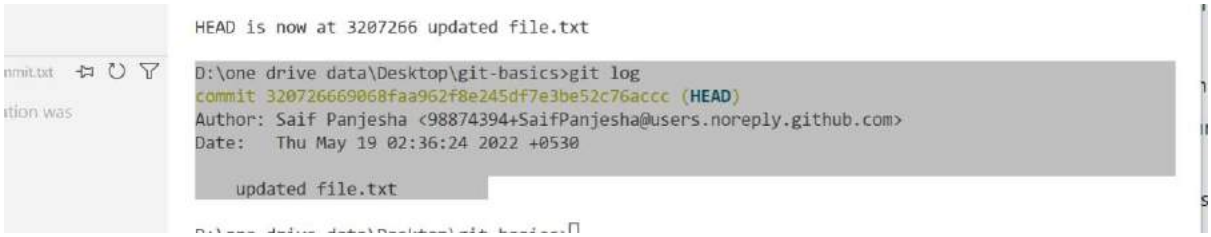


Fig. Not updated the second commit in history

Can the Second Commit is deleted? As we back to previous commit?

The Answer is <u>No</u> not deleted.

To check go to master branch and again check the history.

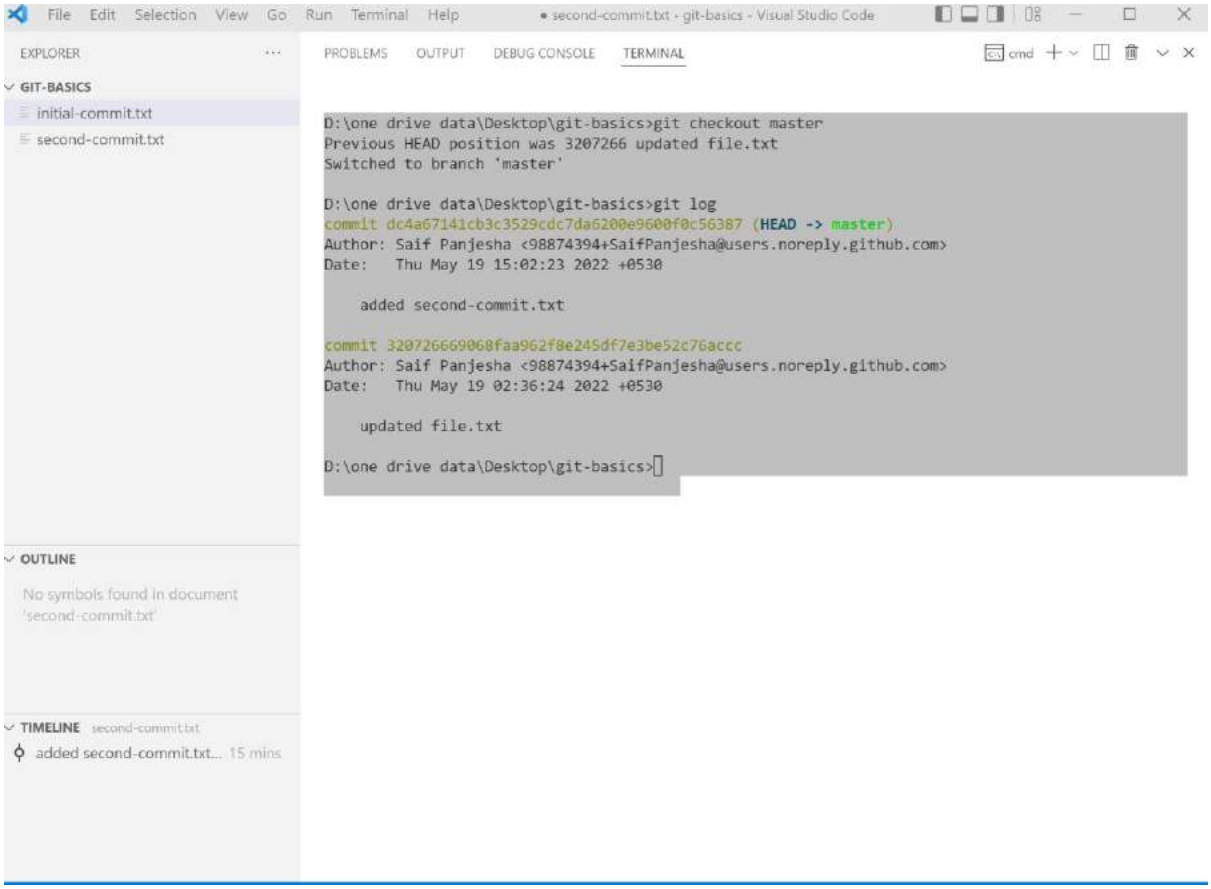


Fig. Second commit is visible.

Understanding and Creating Branches

Branch: is a unique set of specific code changes

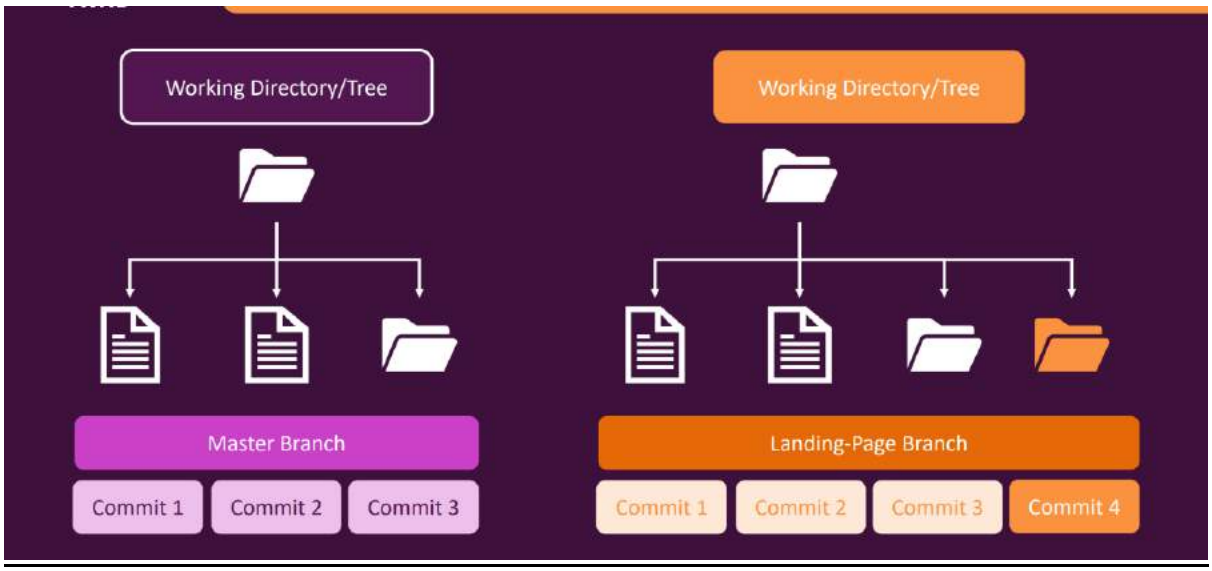


Fig. Goal to create Such copy of master branches

To check of all branches our current project:

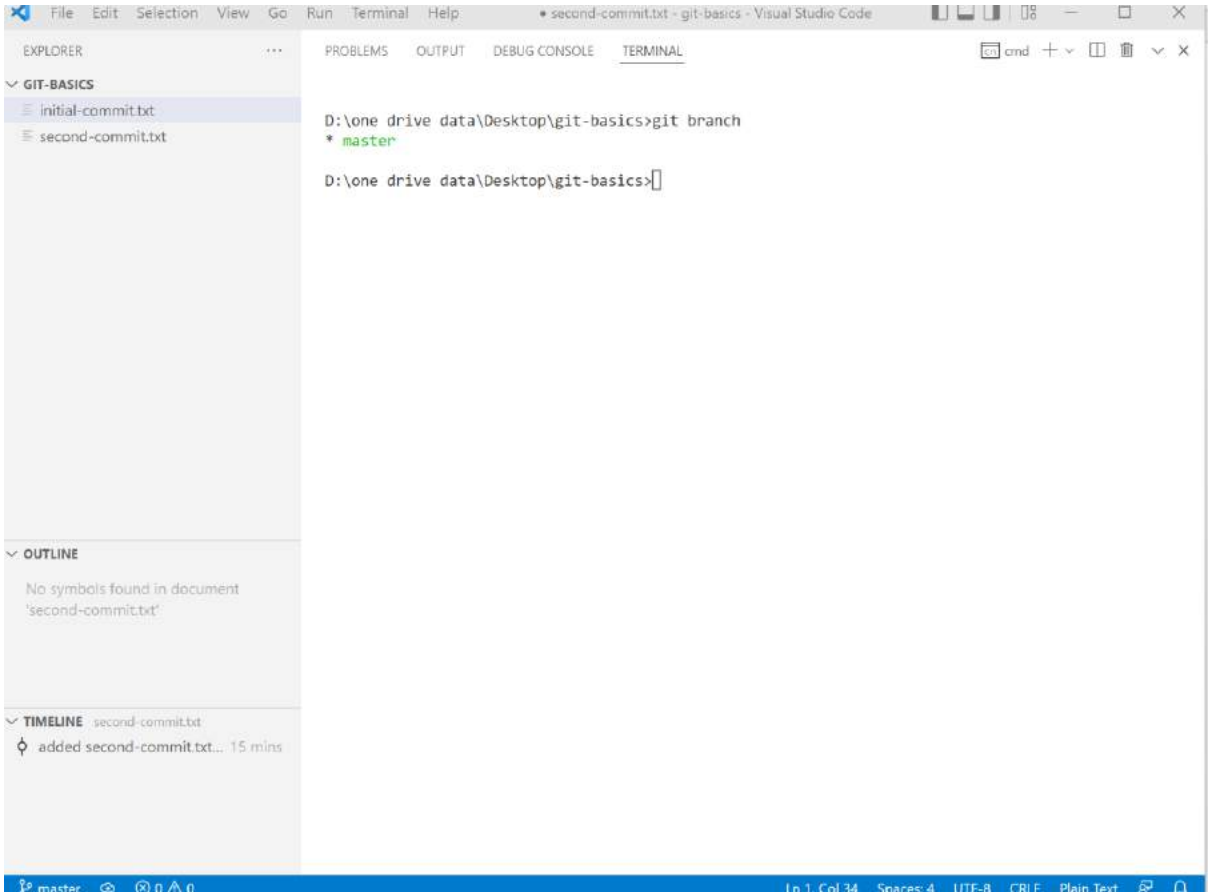


Fig. Default Branch in current project-* master

Creating Second Branch Landing-Page Branch

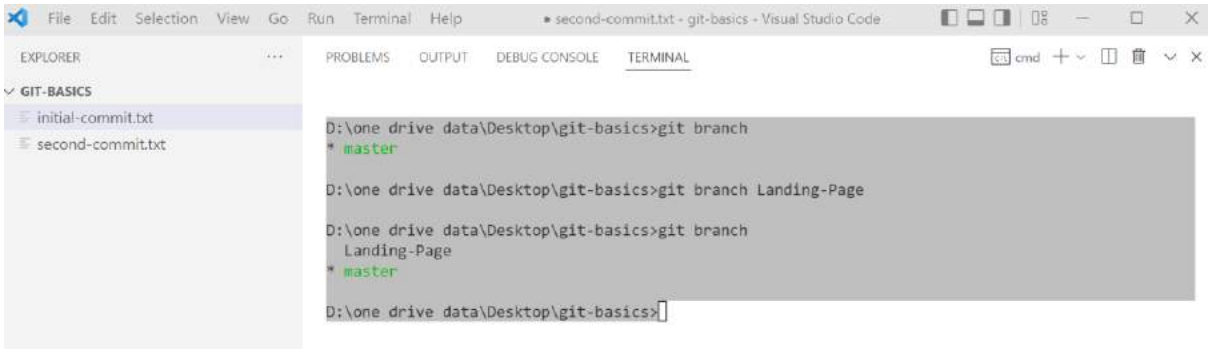


Fig. Second Branch is Created

For accessing Second Branch git checkout branch-name:

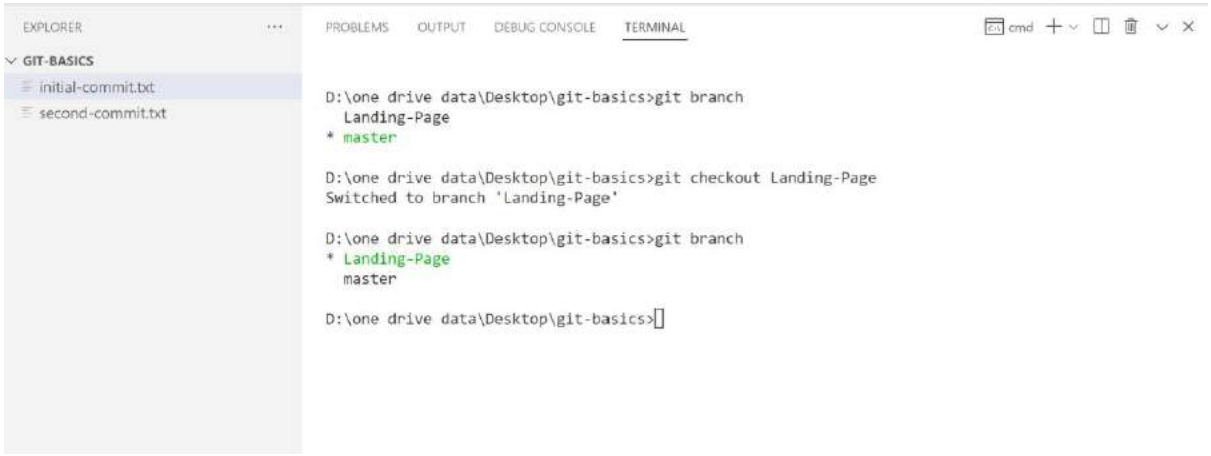


Fig. Accessing Second Branch in git

Shortcut for this type of Operation:

```
D:\one drive data\Desktop\git-basics>git checkout -b quickfix
Switched to a new branch 'quickfix'

D:\one drive data\Desktop\git-basics>git log
commit dc4a67141cb3c3529cdc7da6200e9600f0c56387 (HEAD -> quickfix, master, Landing-Page)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Thu May 19 15:02:23 2022 +0530

    added second-commit.txt

commit 320726669068faa962f8e245df7e3be52c76accc
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Thu May 19 02:36:24 2022 +0530

    updated file.txt
```

Fig. Created third branch

Create new File Working with Branches

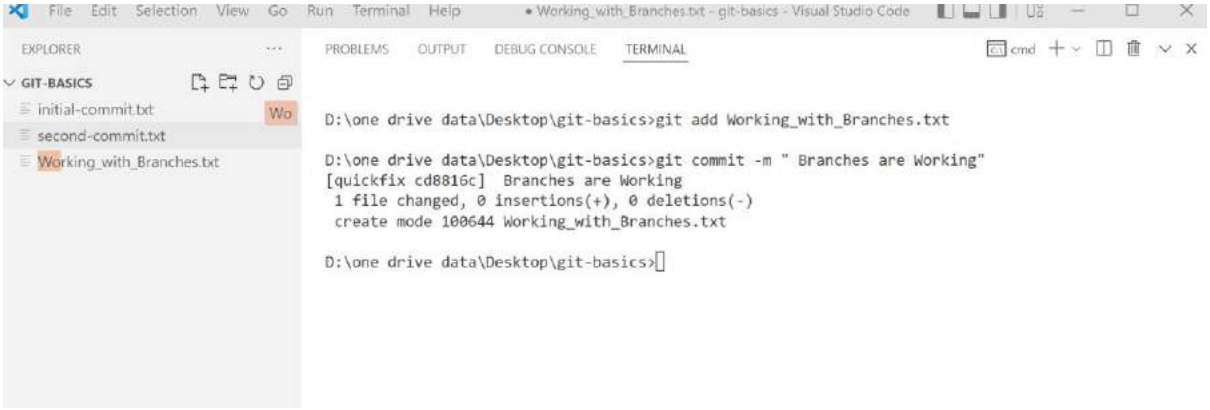


Fig. Working with Branches

```
D:\one drive data\Desktop\git-basics>git log
commit cd8816cdbe74cecbaf4f5995e25b2f9bab3cb9a3 (HEAD -> quickfix)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Thu May 19 15:44:16 2022 +0530

    Branches are Working

commit dc4a67141cb3c3529cdc7da6200e9600f0c56387 (master, Landing-Page)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Thu May 19 15:02:23 2022 +0530

    added second-commit.txt

commit 320726669068faa962f8e245df7e3be52c76accc
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Thu May 19 02:36:24 2022 +0530

    updated file.txt
```

Fig. Heading with third branch

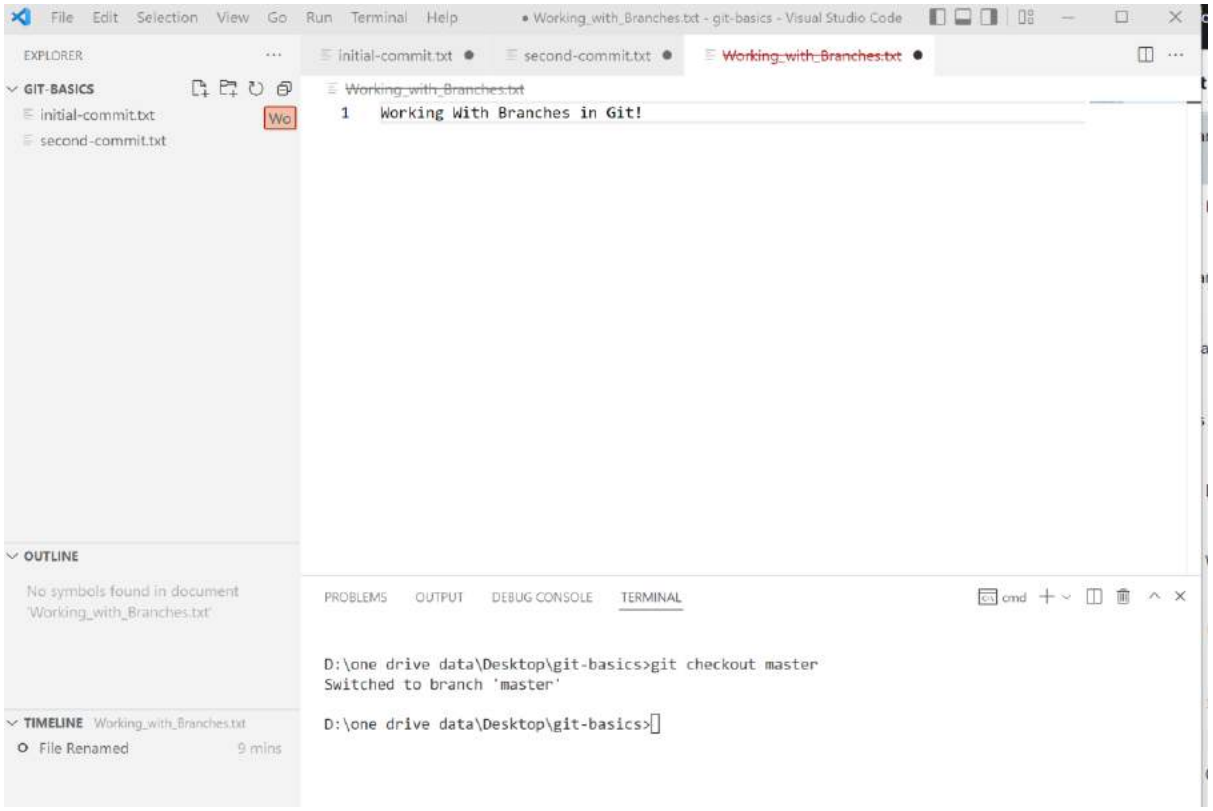


Fig. Switch to Master Branch

After Switch to master branch? Does it reflect the Working with Branches in third branch (quickfix)?How to merge this branches ??

Merging Branches – The Basics

D:\one drive data\Desktop\git-basics>git merge quickfix

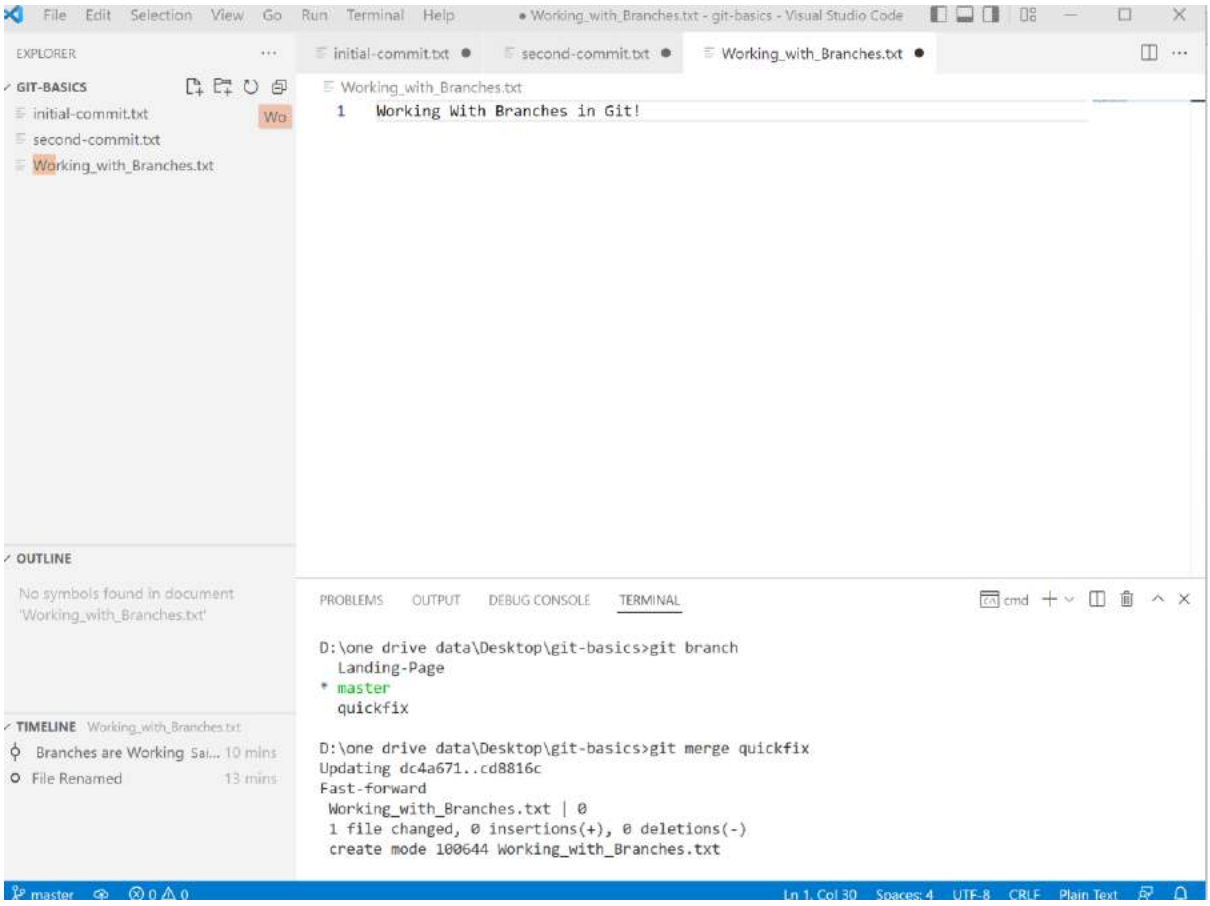


Fig. Merging with Branch

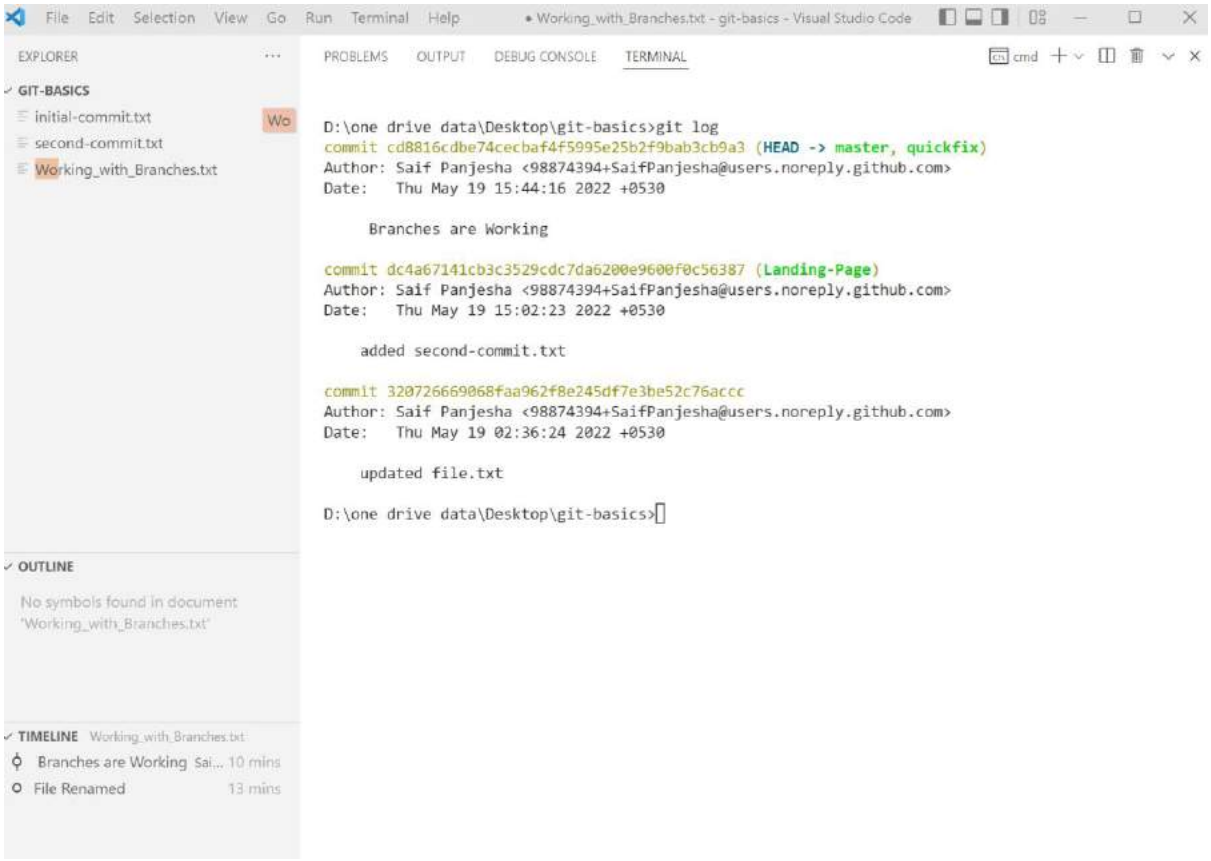


Fig. Success History of Merging.

Understanding the HEAD

The Latest commit is known as HEAD.

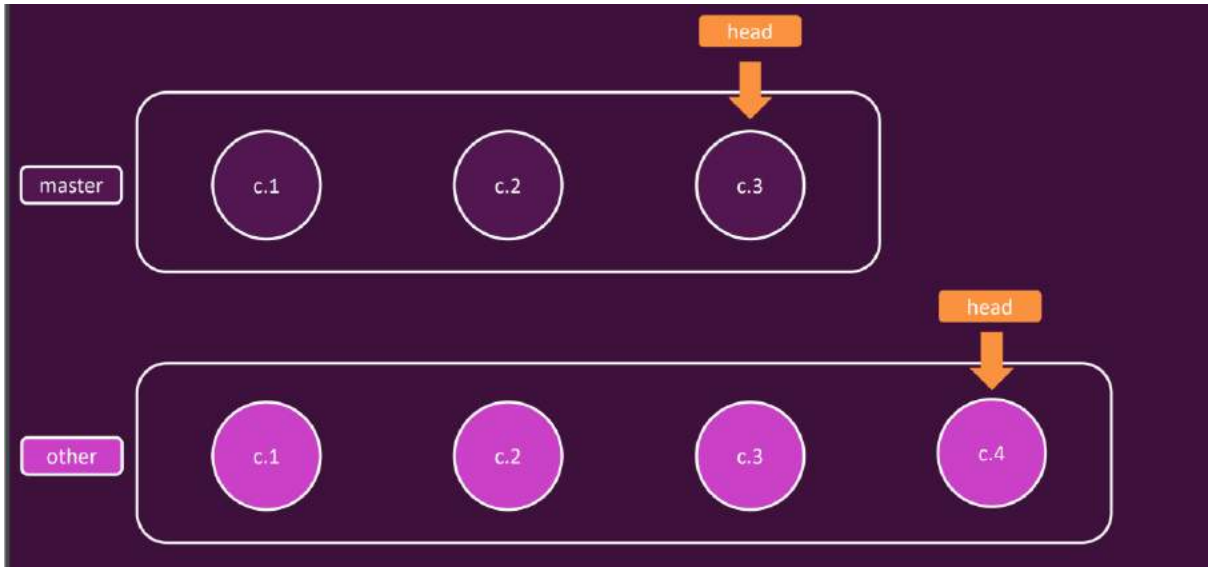


Fig. Head Understanding

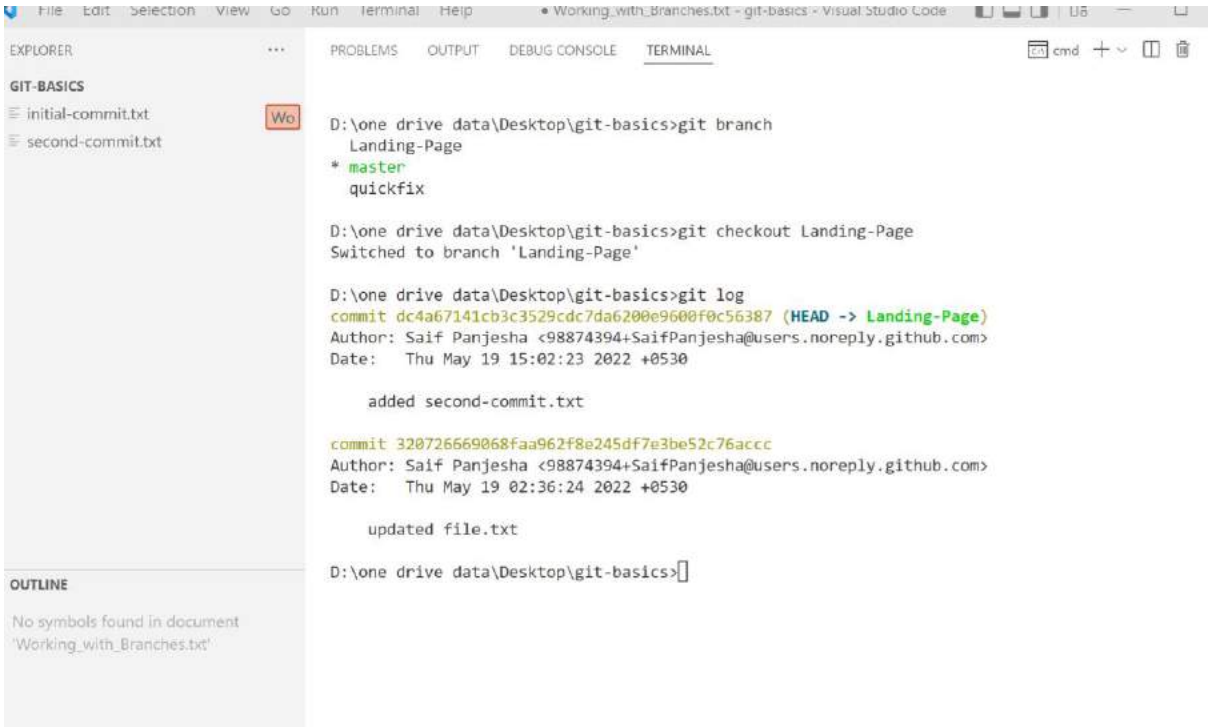


Fig. Head Understanding History

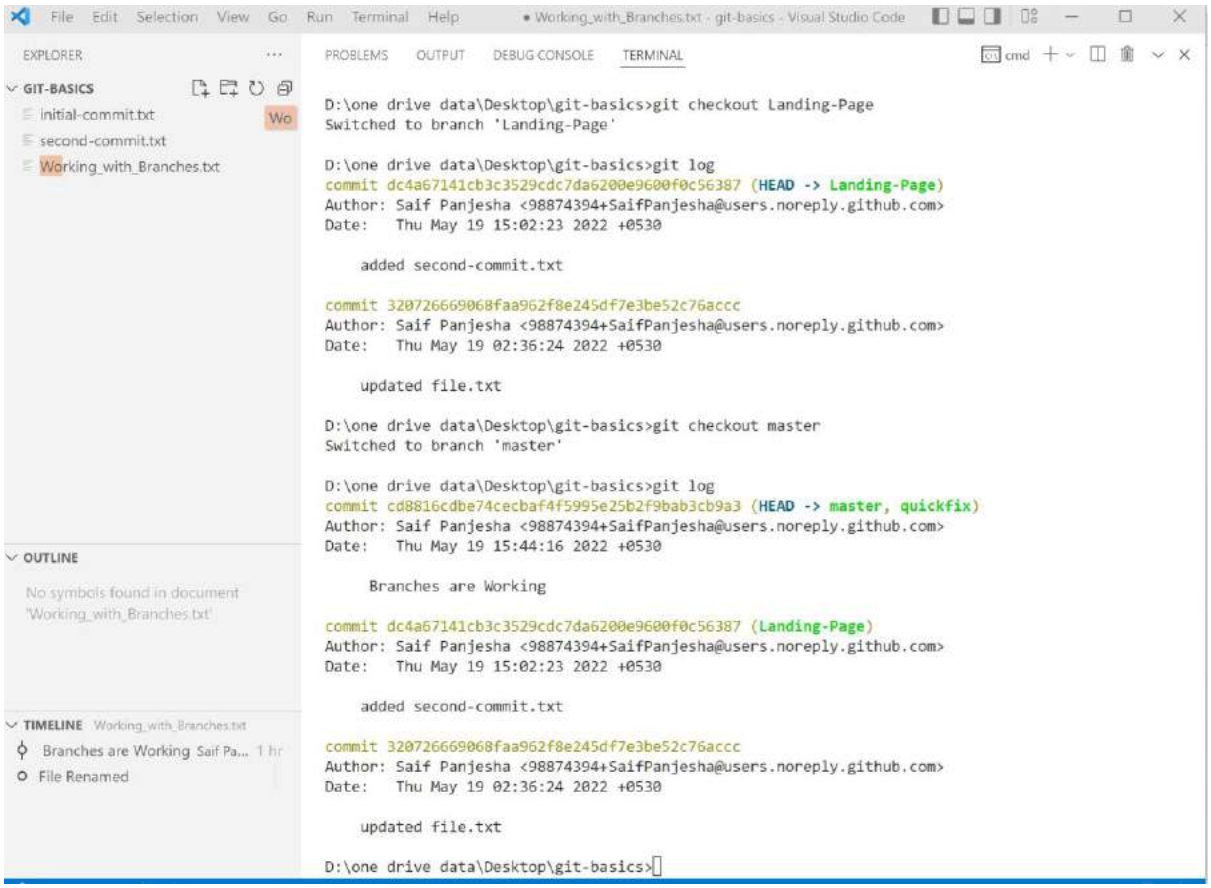


Fig. Head Understanding with Latest Commit on History

Detached Head

What is Detached Head?

Detached HEAD indicates that the currently checked-out repository is not a local branch. This can be caused by the following scenarios:

- When a branch is a read-only branch and we try to create a commit to that branch, then the commits can be termed as “free-floating” commits not connected to any branch. They would be in a detached state.
- When we checkout a tag or a specific commit and then we try to perform a new commit, then again the commits would not be connected to any branch. When we now try to checkout a branch, these new commits would be automatically placed at the top

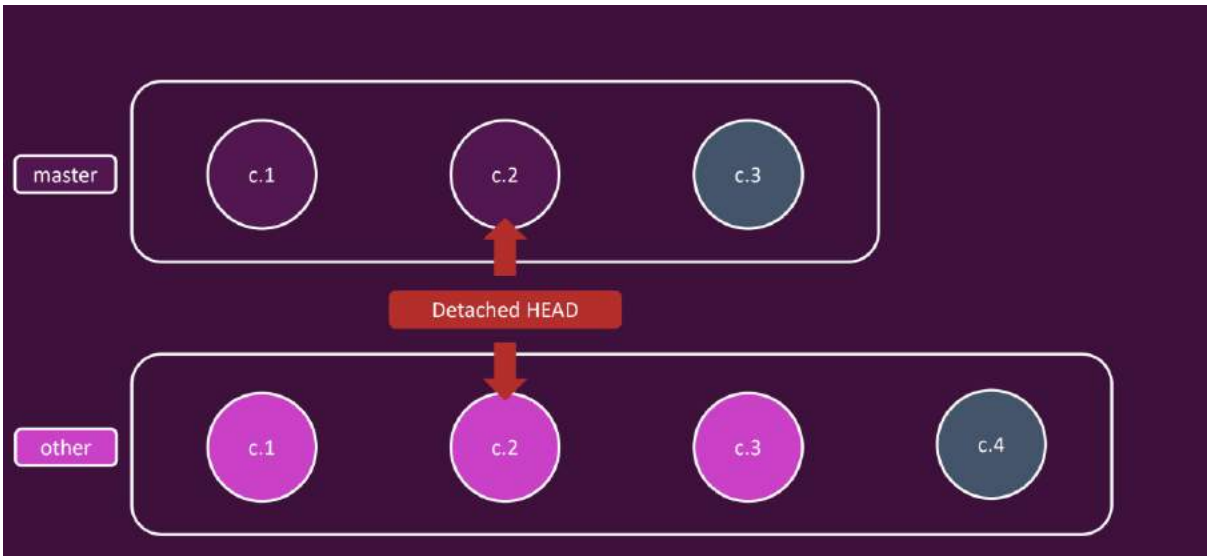


Fig. Detached Head

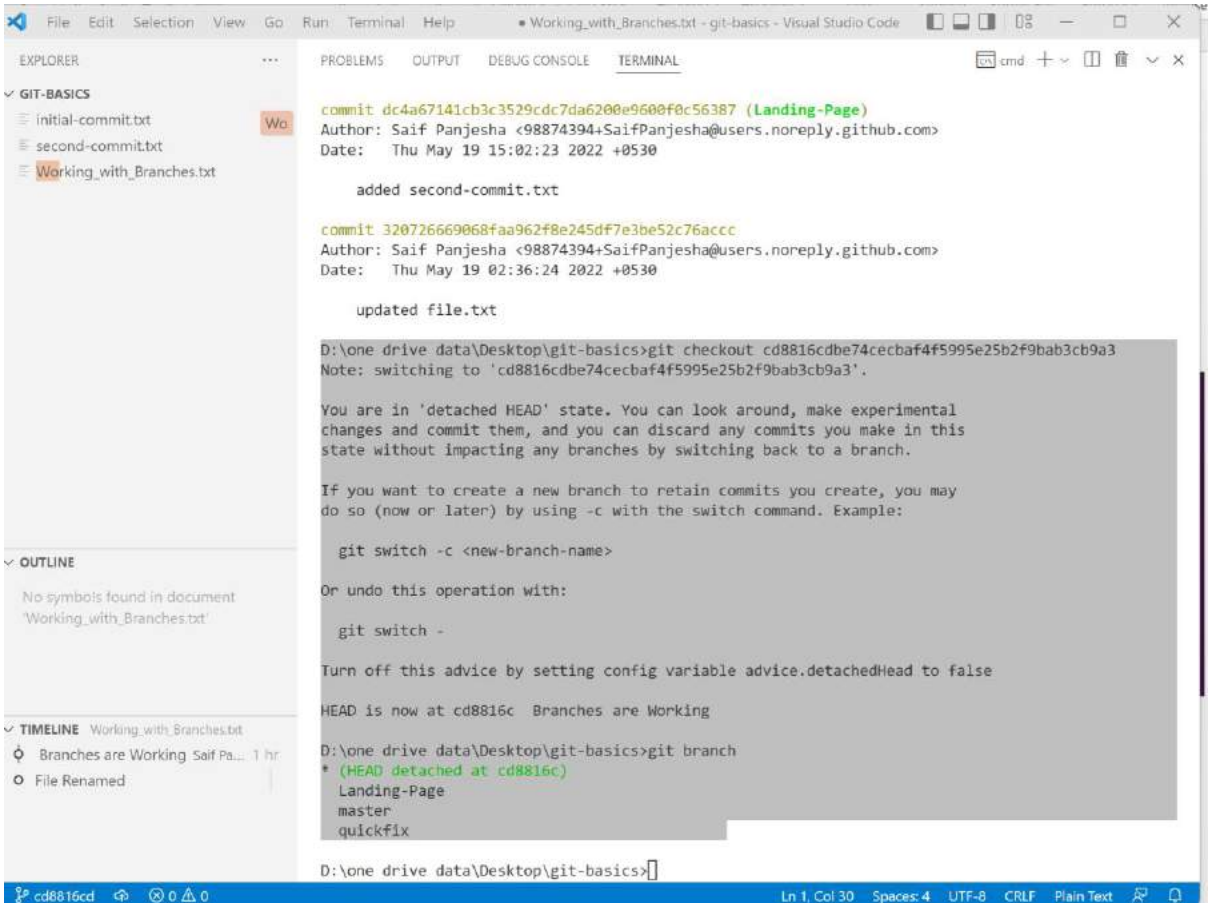


Fig. Detached Head on terminal

How to avoid this?

In order to ensure that detached state doesn't happen, =instead of checking out commit/tag, we can create a branch emanating from that commit and then we can switch to that newly created branch by using the command:

git checkout <<branch_name>. This ensures that a new branch is checkout out and not a commit/tag thereby ensuring that a detached state wouldn't happen

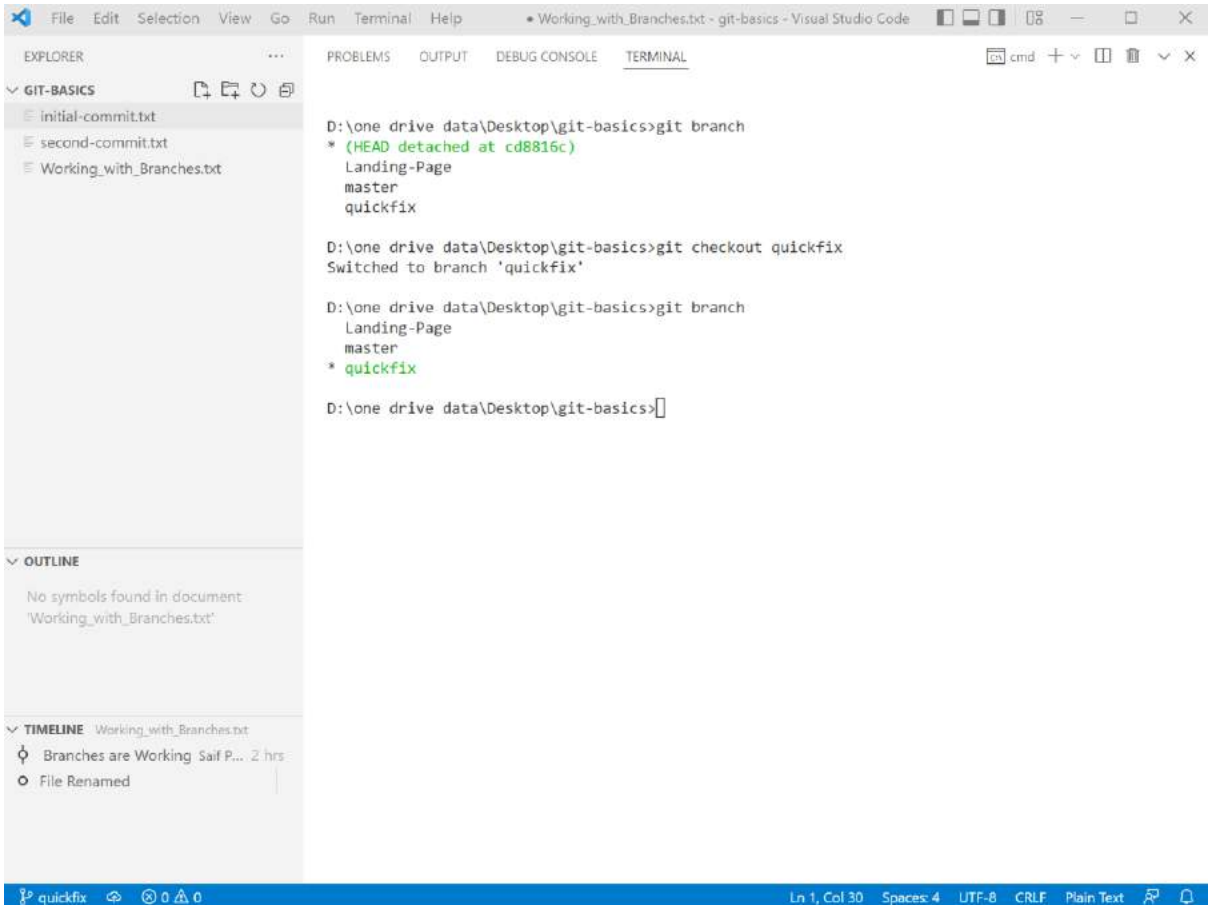


Fig. Avoided Detached Head

Branches & “git switch” (Git 2.2.23)

The “git switch” command

The switch command allows you to do , well, switch branches and create new branches. Now you would say, "Why would I want to have a new command in here?" Well the idea basically is that checkout can be used for commits and for branches so it can be confusing, especially for beginners. So with switch is the nice shortcut for creating a new branch.

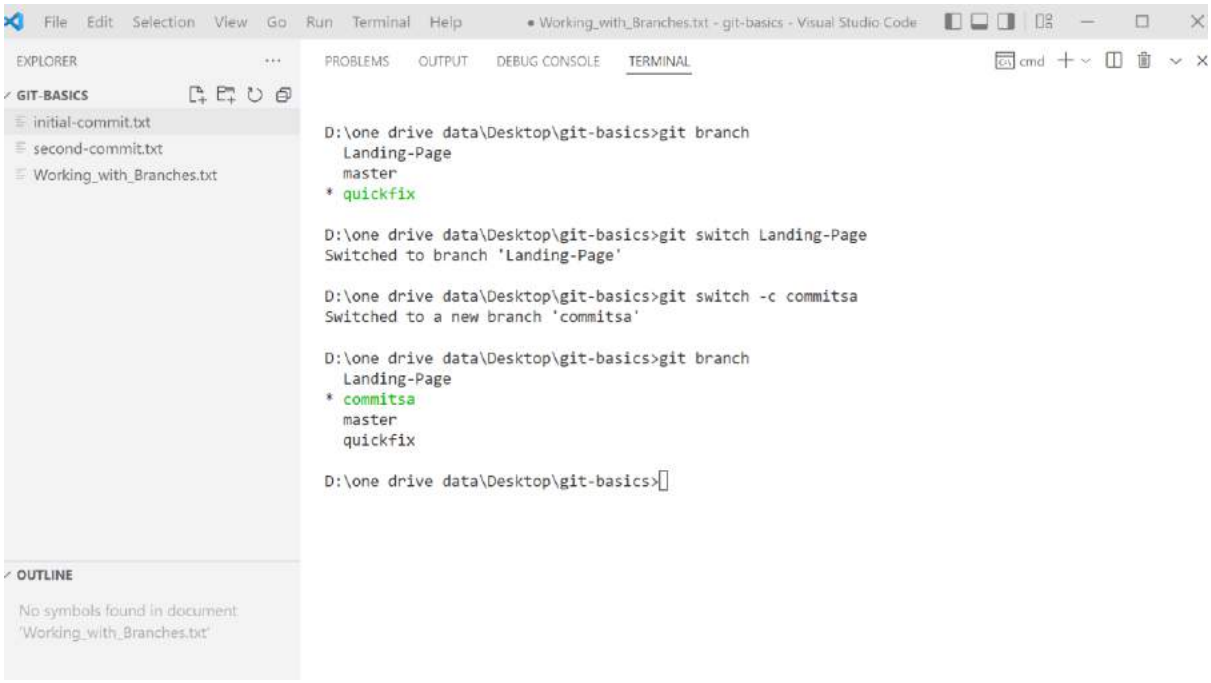


Fig. Created new branch with git switch

How to reserve the Steps and fix when something wrong with working directory, Branches & Commit:

Fig. Deleting Data : Overview

Deleting Working Directory Files:

To check Which files currently on staging area.

git ls-files

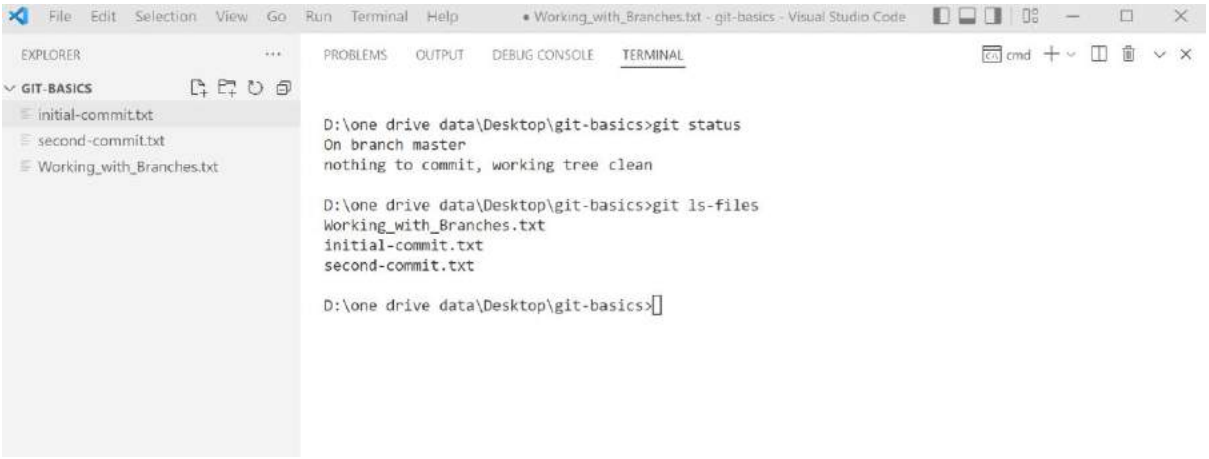


Fig. show the staging area of master branch master

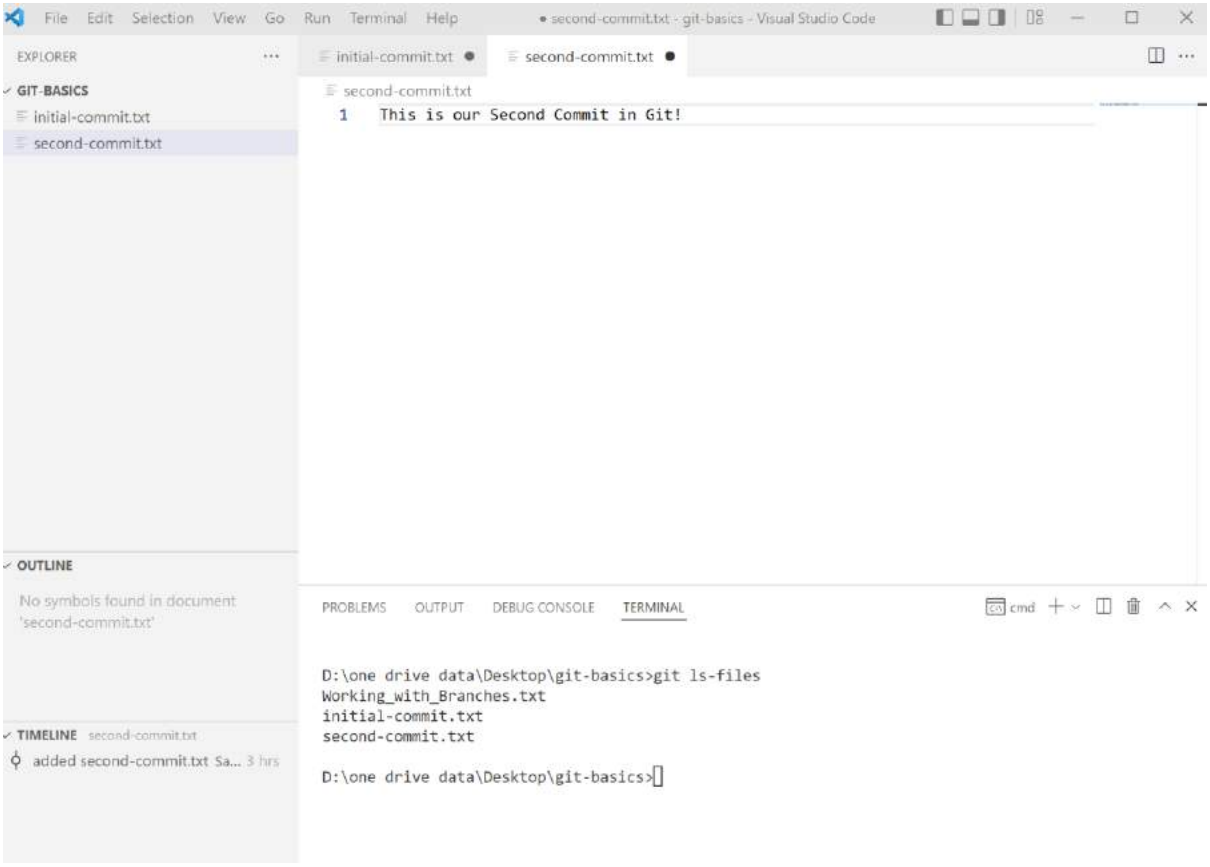


Fig. After deletion of file still show on staging area.

How to remove the file which has been deleted from working directory? In staging area??

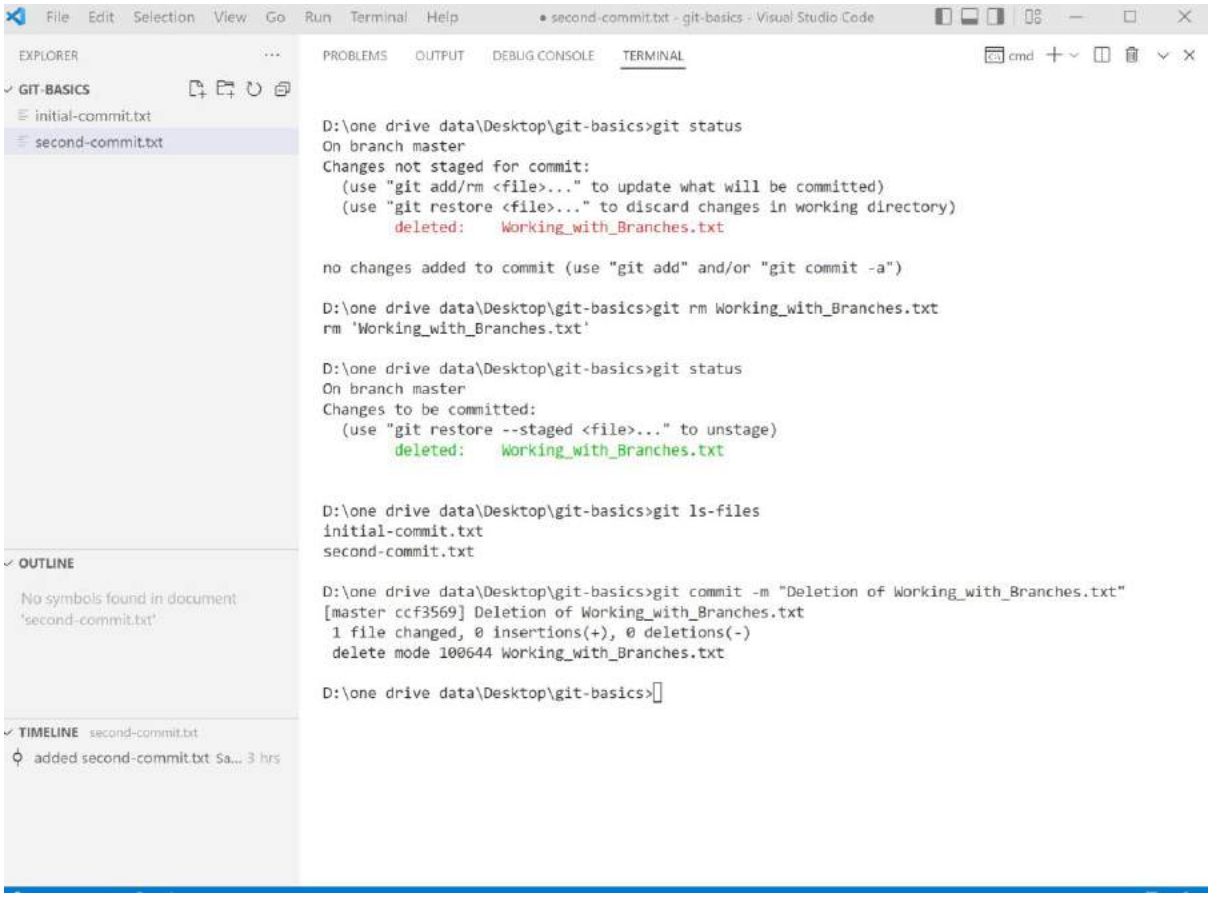


Fig. Delete successful from staging area

Undoing Unstaged Changes?

Added Extra Information not needed on Initial -Commit .txt

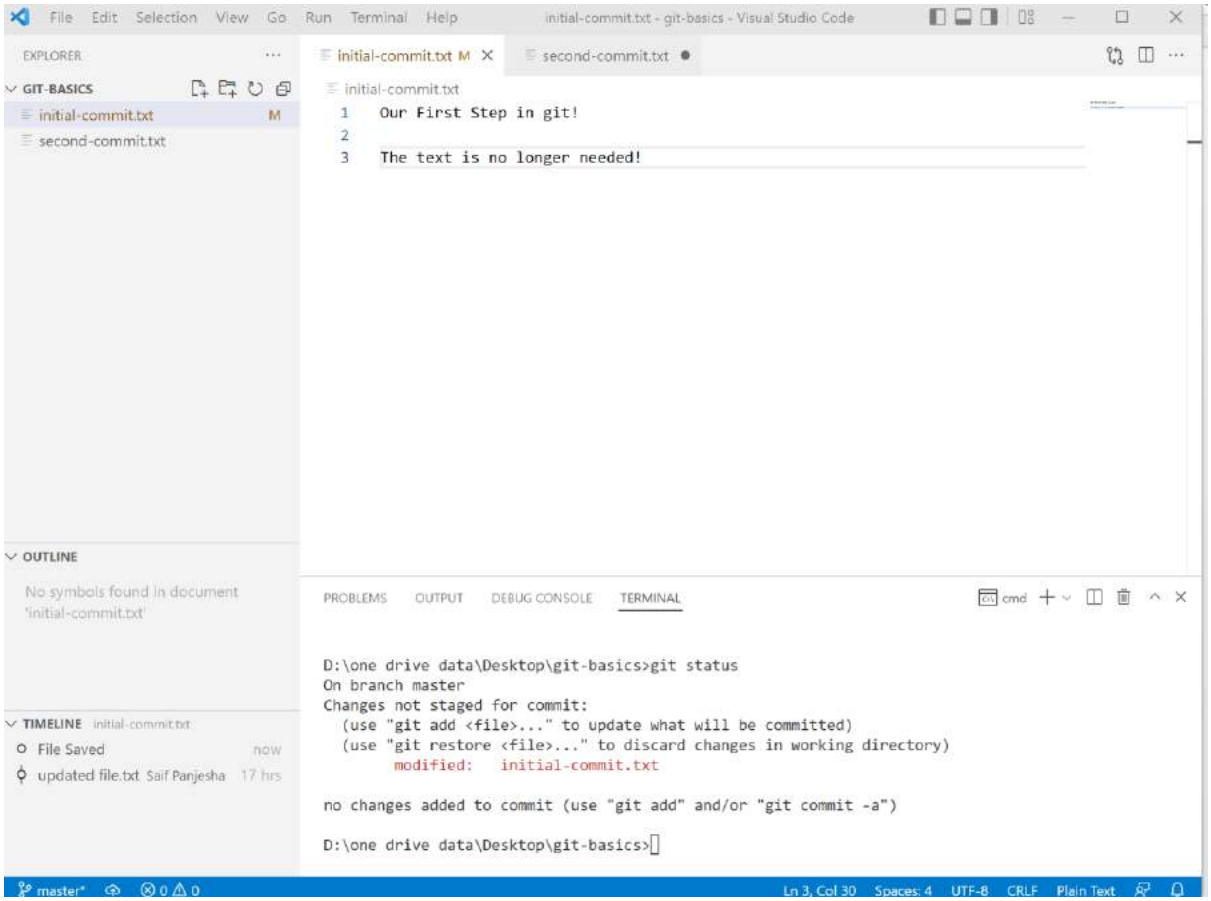


Fig. Unstaged Changes

How to go back from these changes??

git checkout initial-commit.txt

```
D:\one drive data\Desktop\git-basics>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   initial-commit.txt

no changes added to commit (use "git add" and/or "git commit -a")

D:\one drive data\Desktop\git-basics>git checkout intial-commit.txt
error: pathspec 'intial-commit.txt' did not match any file(s) known to git

D:\one drive data\Desktop\git-basics>git checkout initial-commit.txt
Updated 1 path from the index

D:\one drive data\Desktop\git-basics>git status
On branch master
nothing to commit, working tree clean

D:\one drive data\Desktop\git-basics>
```

Fig. Undoing Unstaged Changes

Latest Command for Undoing Unstaged Changes After (Git 2.2.23)

git restore: latest command for Undoing Unstaged Changes after git 2.2.23

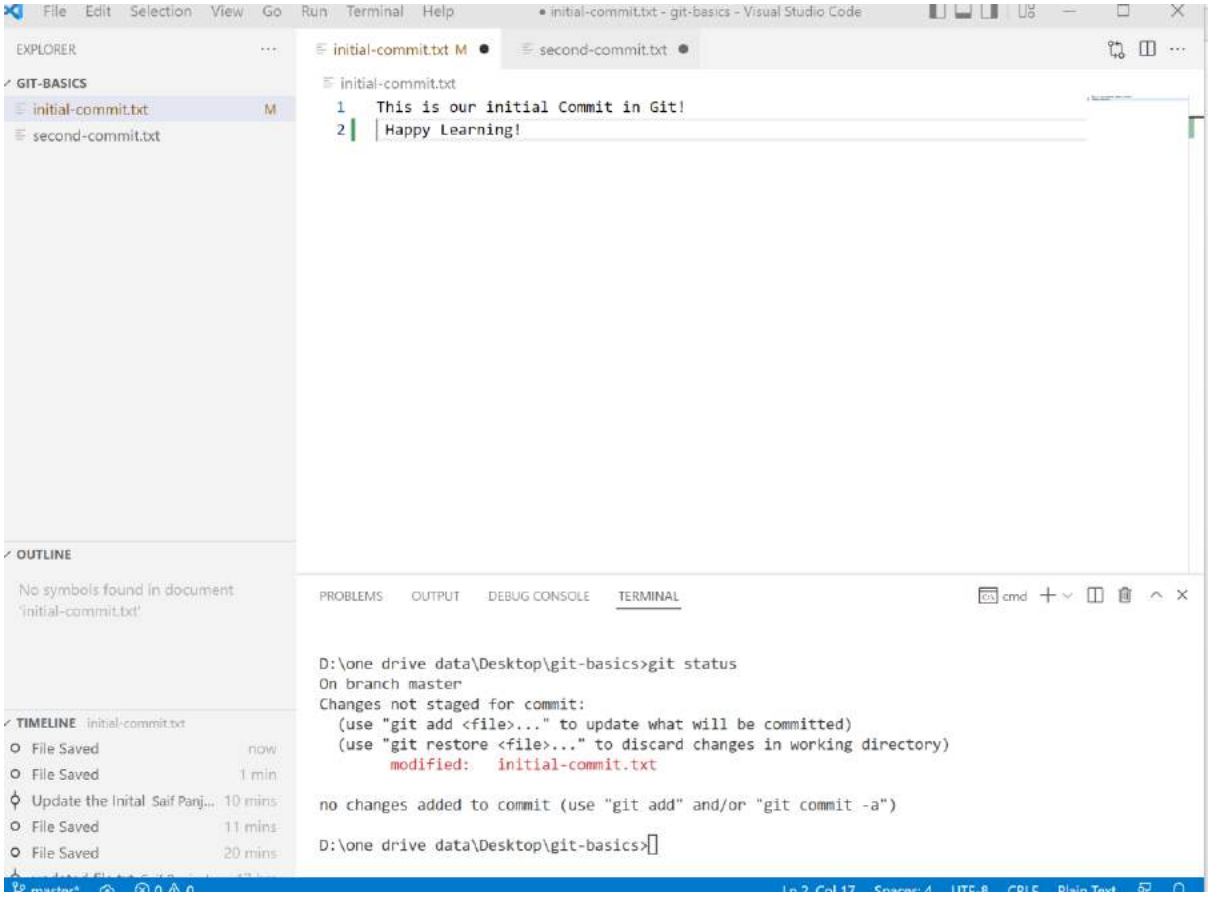


Fig. Unstaged Changes Happens

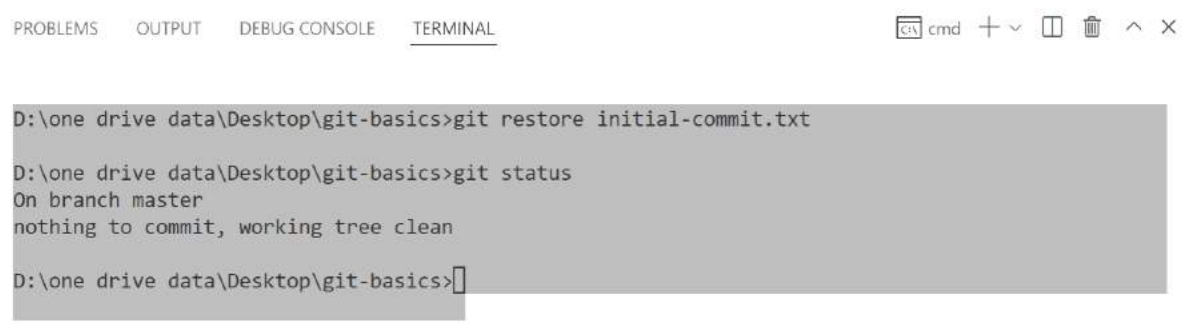


Fig. Successful restore back the changes

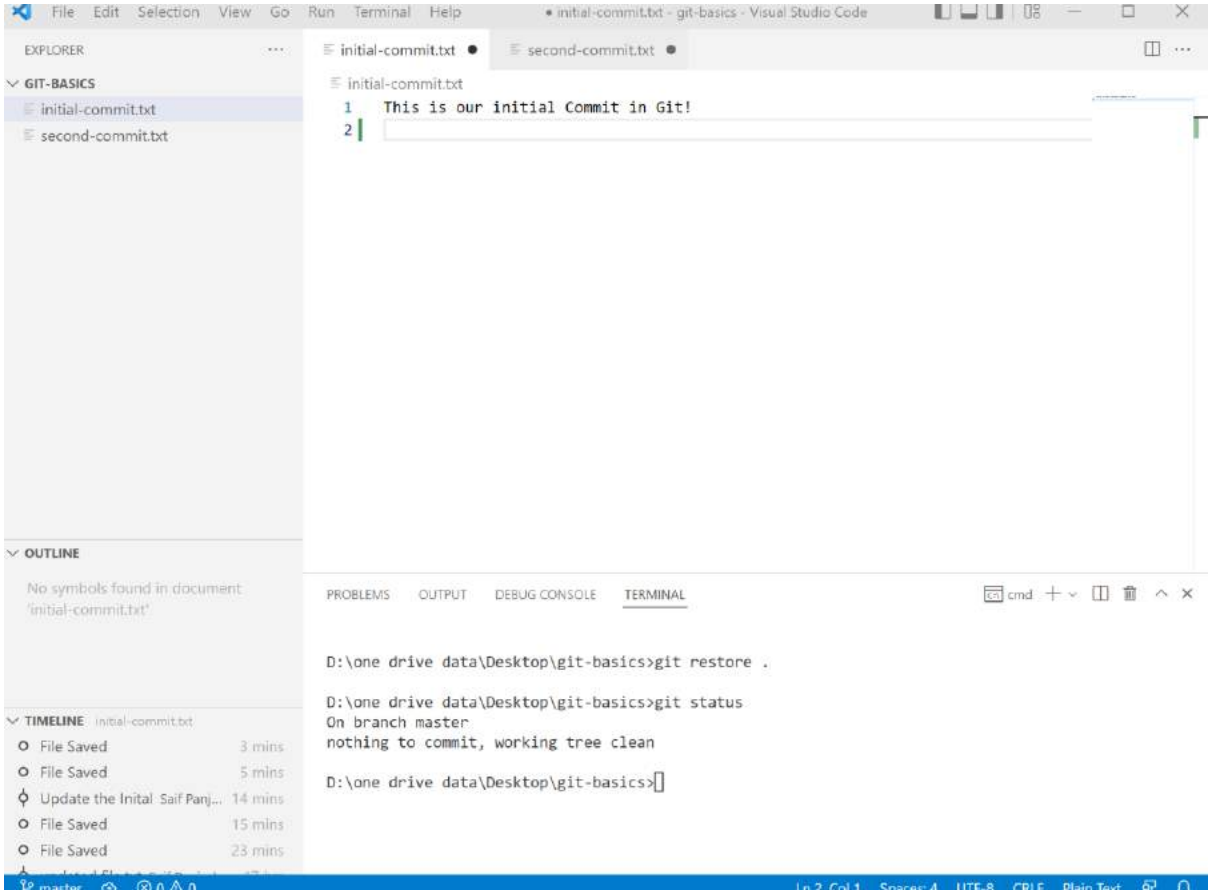


Fig. Successful restore back the changes on multiple statement

How to restore Unstaged File:

“git clean -dn” & “git clean -df”: - remove files or remove force file

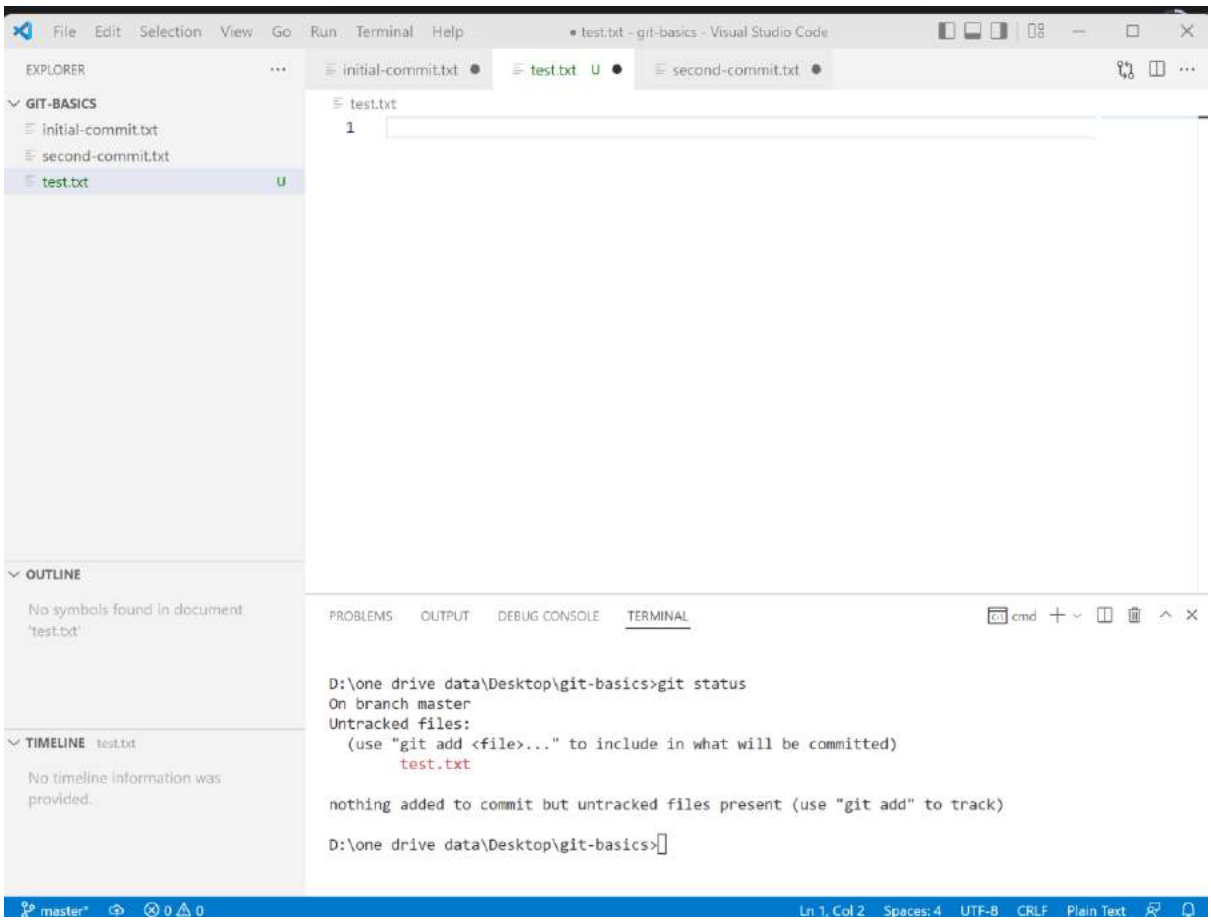


Fig Created file test.txt

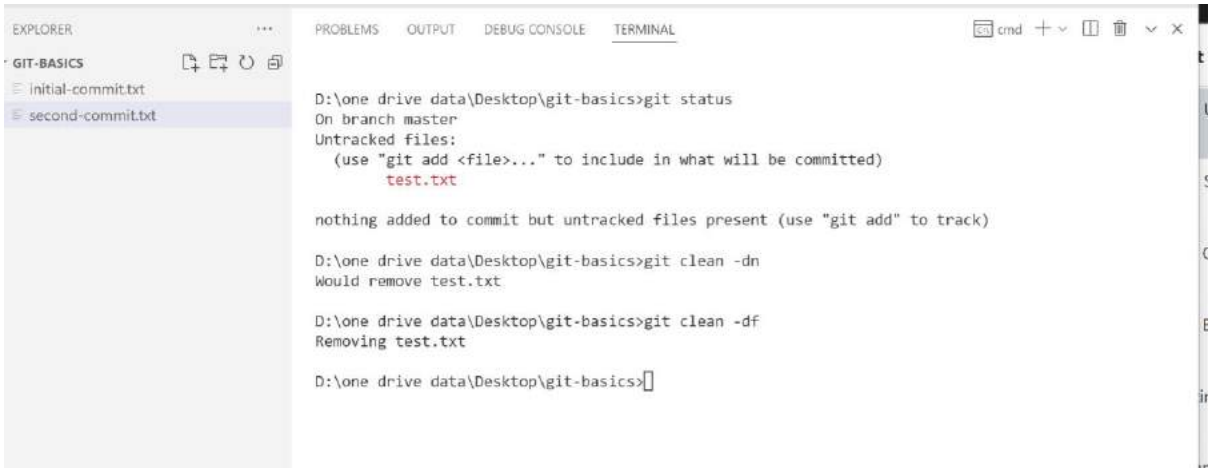


Fig. Remove test.txt

Undoing Staged Changes:

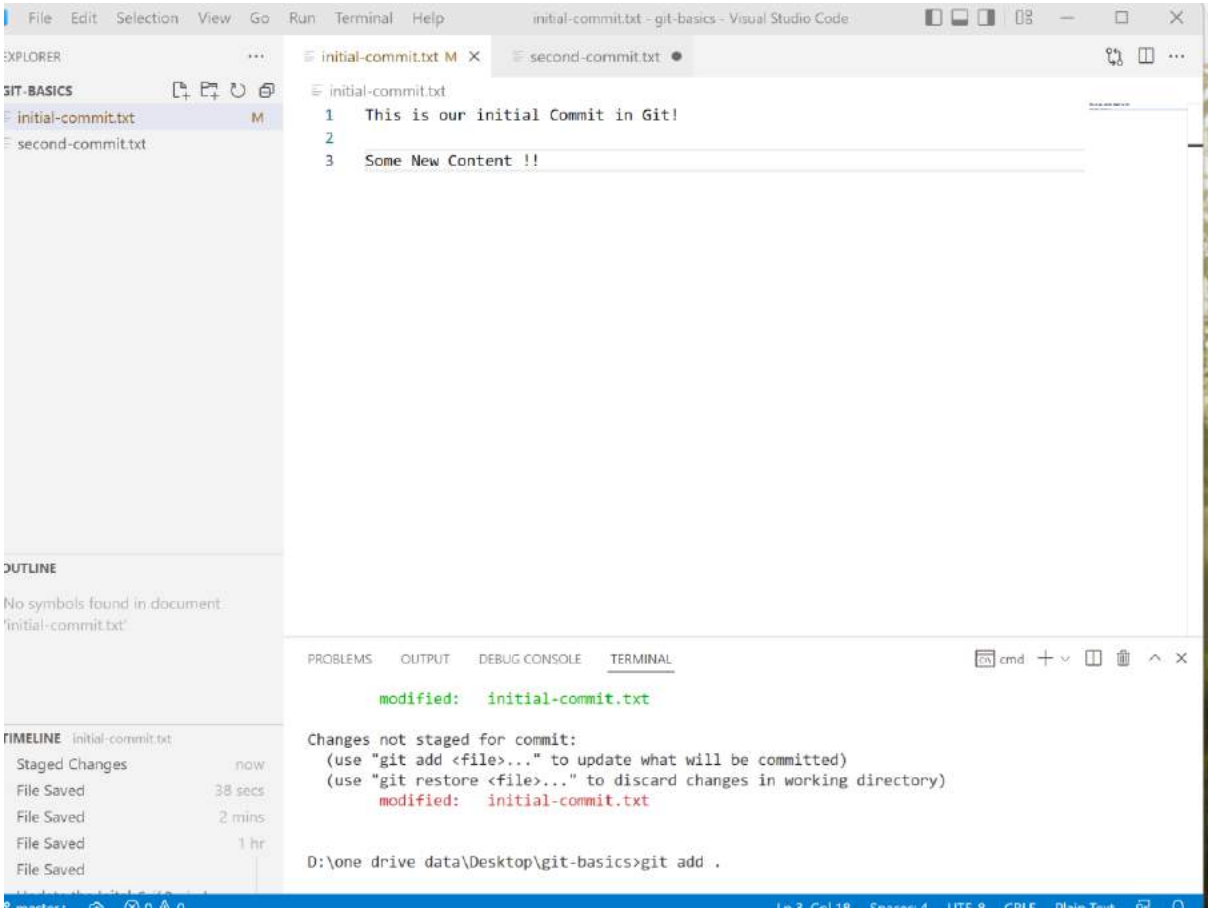


Fig. Added Some Content

Fig. Modified git status

git reset: will help you to bring latest status of commit in staging area and later we can undo the staged changes.

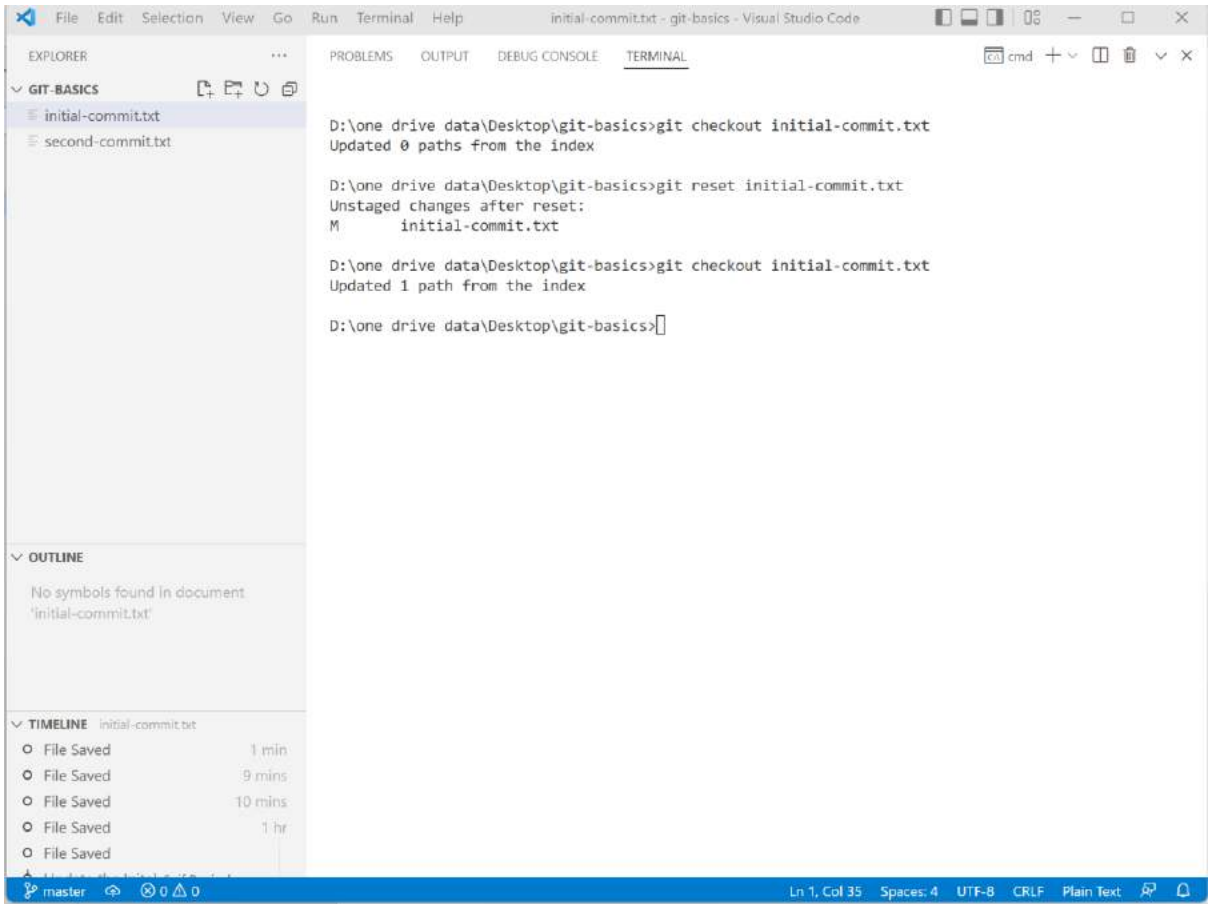


Fig. Staging Changes reset

git restore: will act as same as “git reset” after version Git 2.2.23

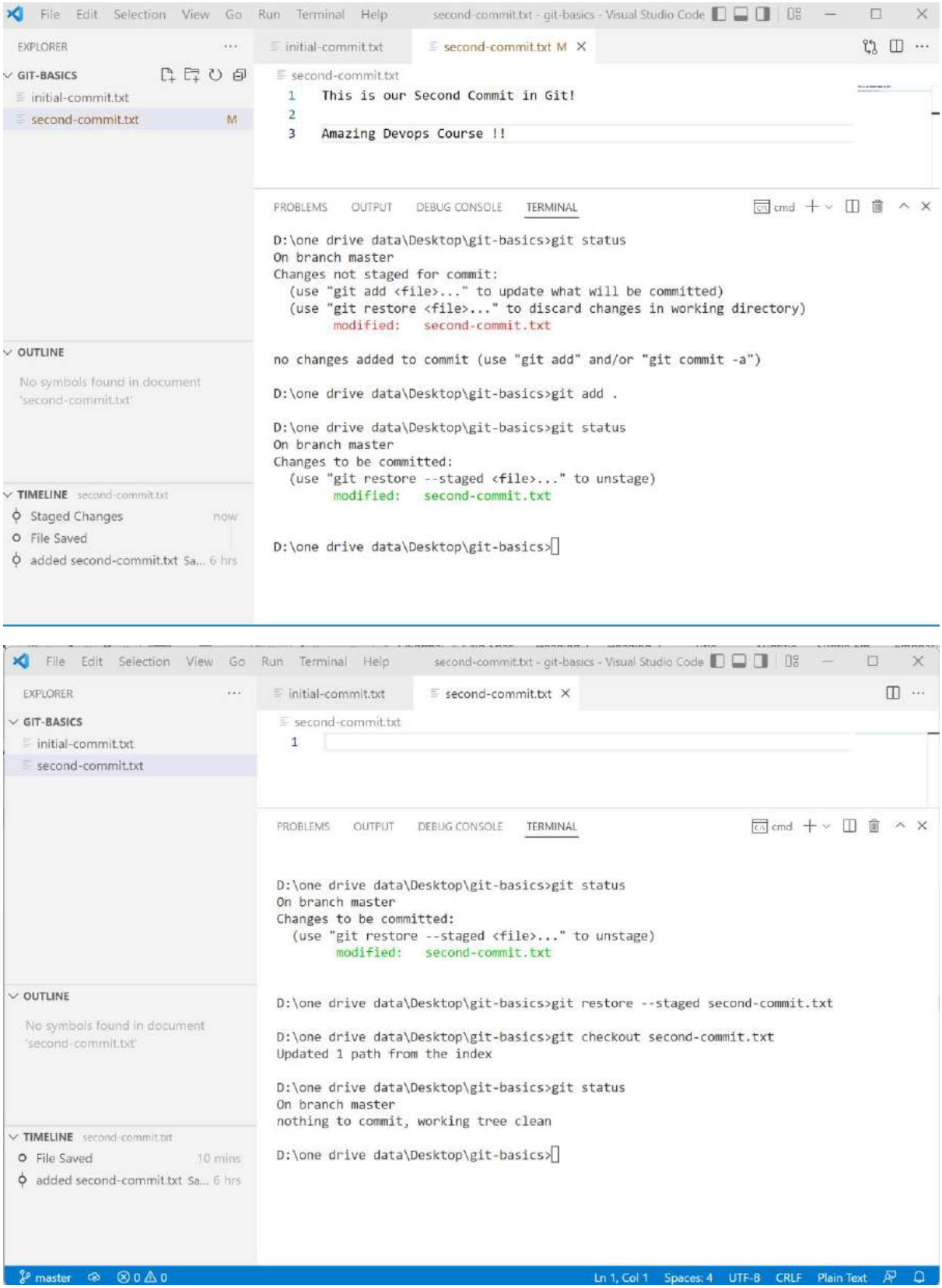


Fig. Success Staged Changes restore

Deleting commit with git reset

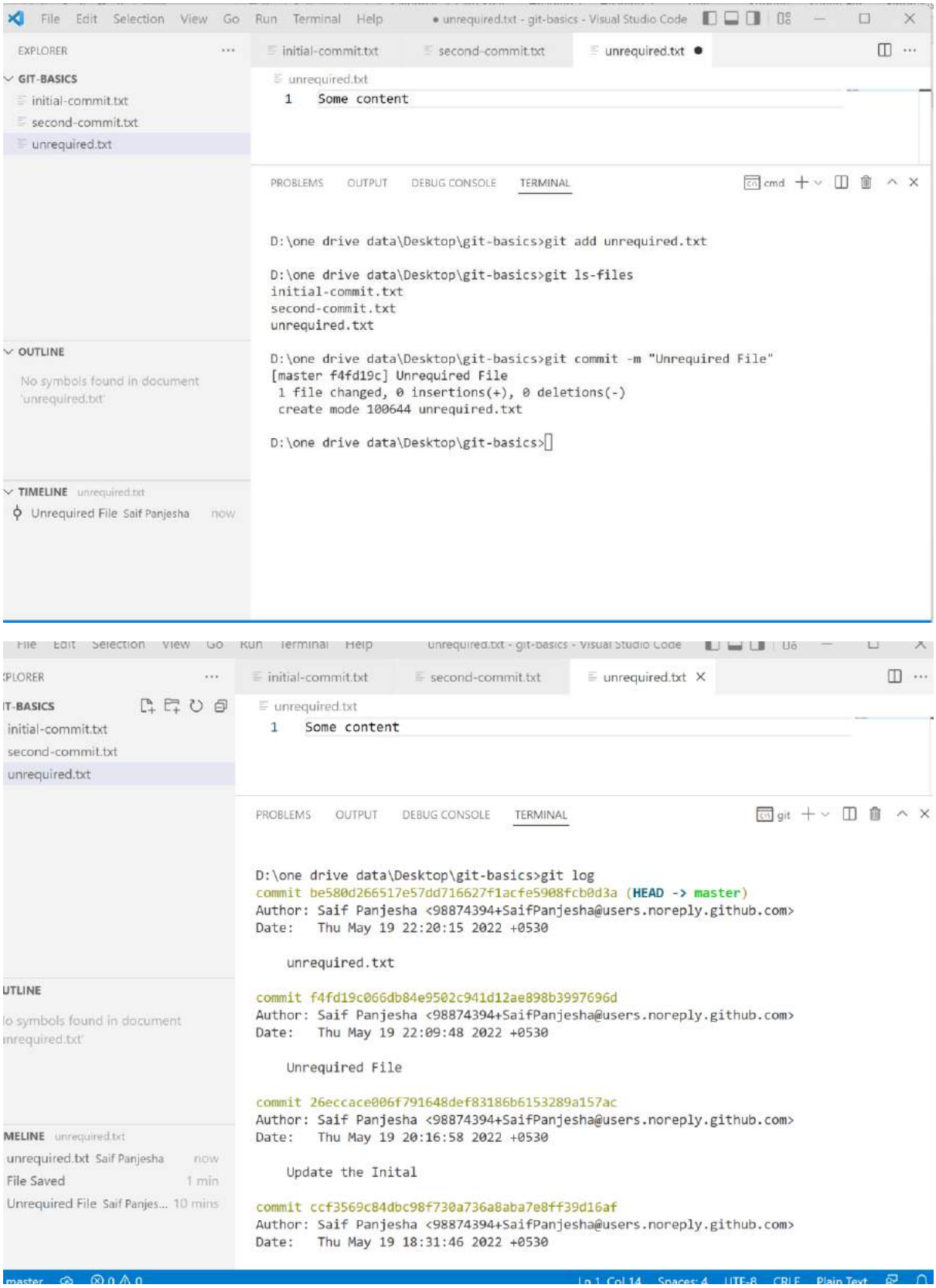


Fig. Logs Heading at Master staging area

As fig show I want to go previous head the update the Initial Commit done by "git reset" with "- -soft" as soft reset

D:\one drive data\Desktop\git-basics>git reset --soft HEAD~2

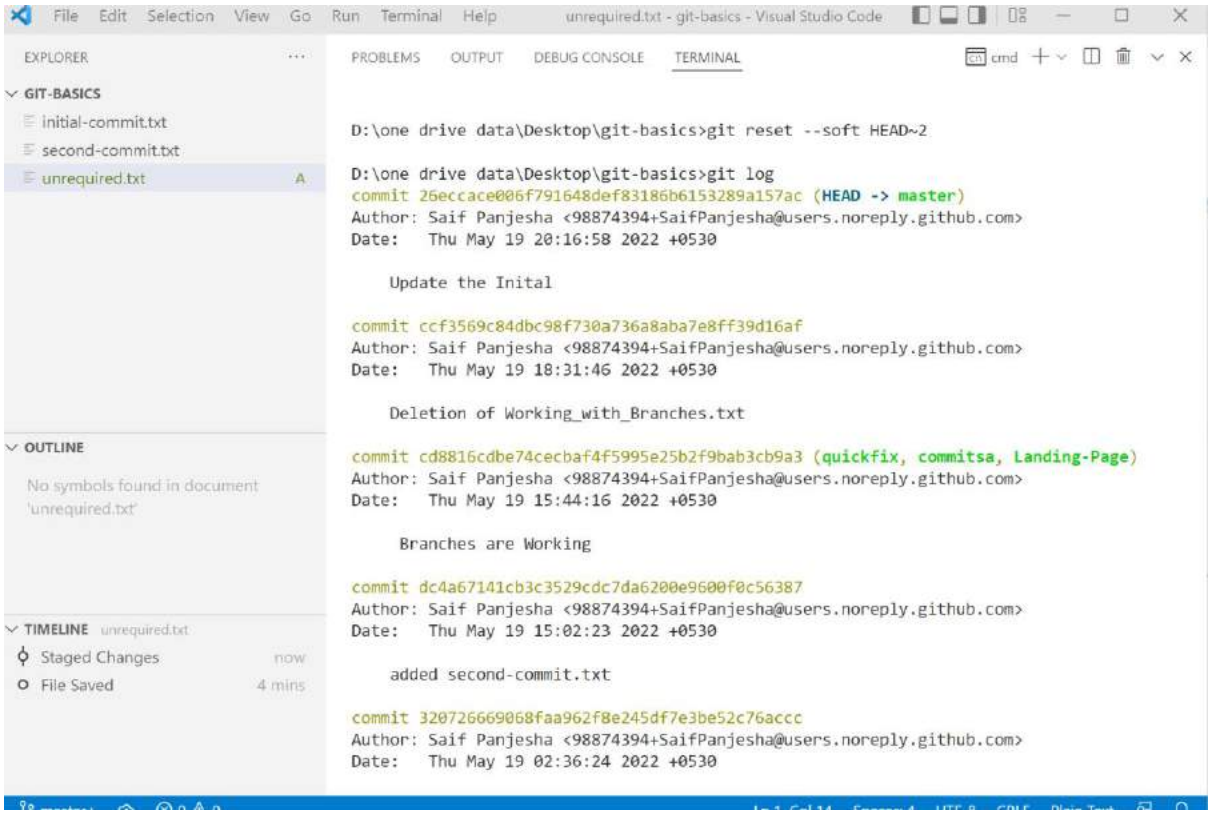


Fig. Head -> master to Update the Initial state

Default Command: git reset HEAD ~2

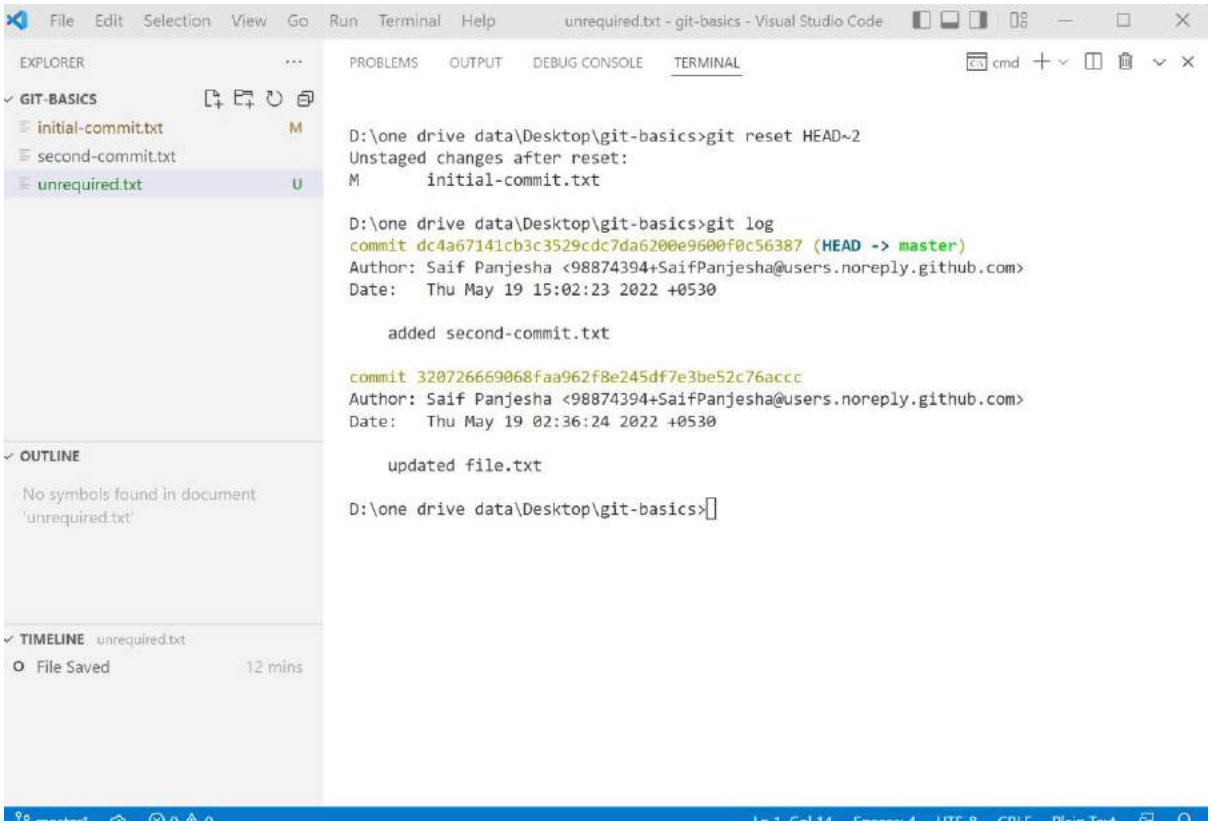


Fig. Head -> master to added second-commit.txt

Successful Deleting file:

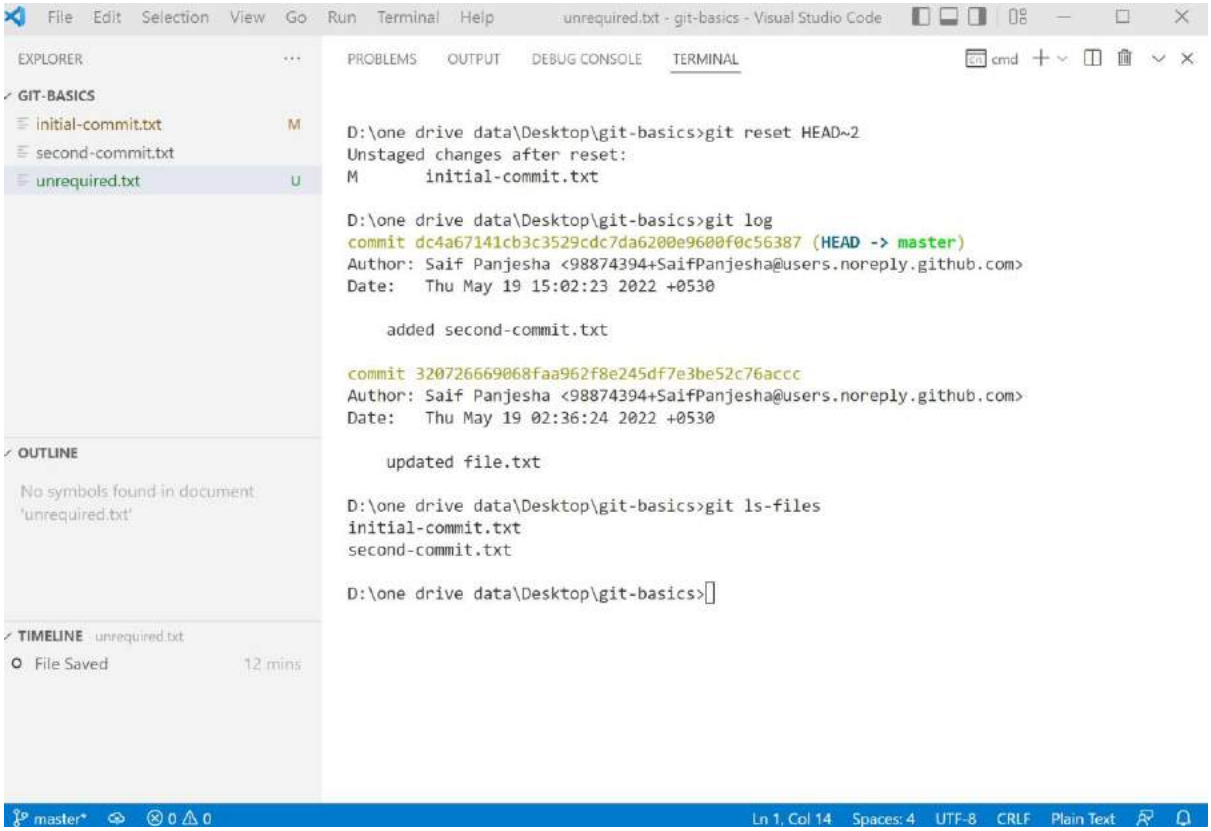


Fig. Success Deletion on unrequired file

Fig. Added file again

“git reset” with “- -hard” as Hard reset

Removing all changes from Working directory & Staging area

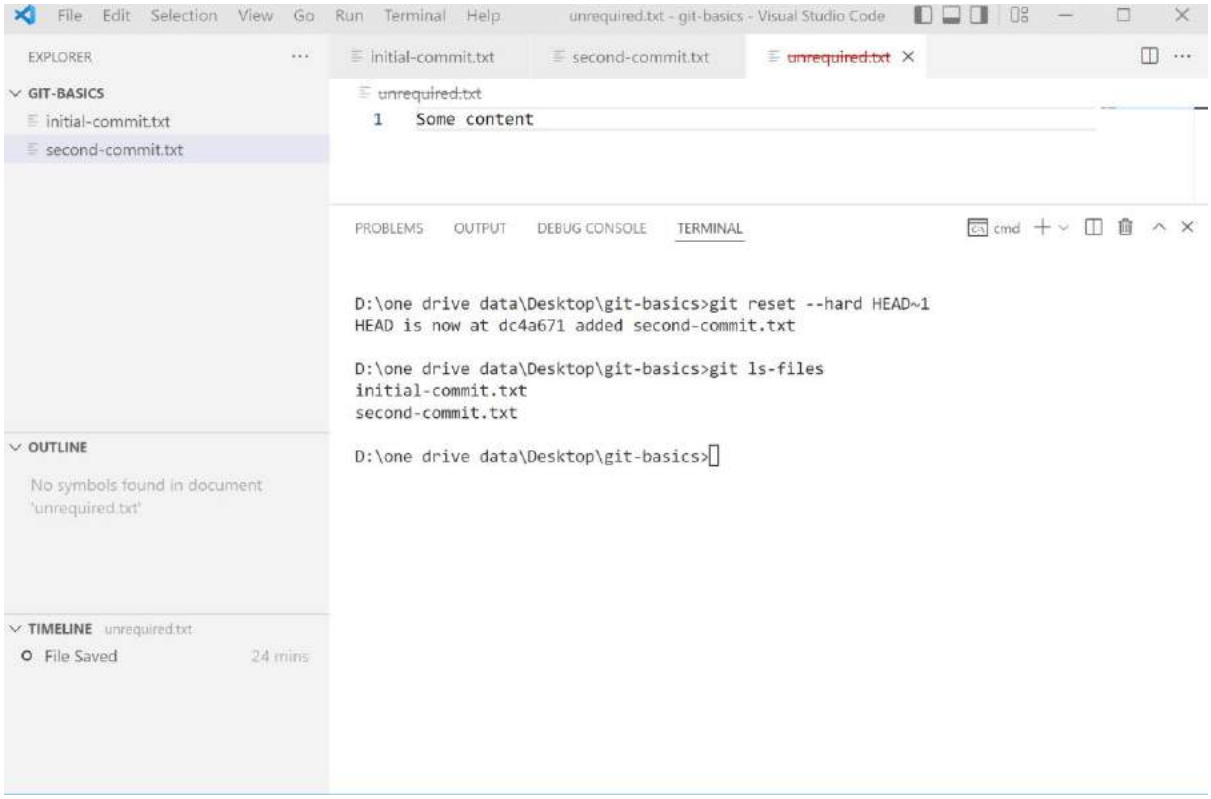


Fig. - - hard removing all area

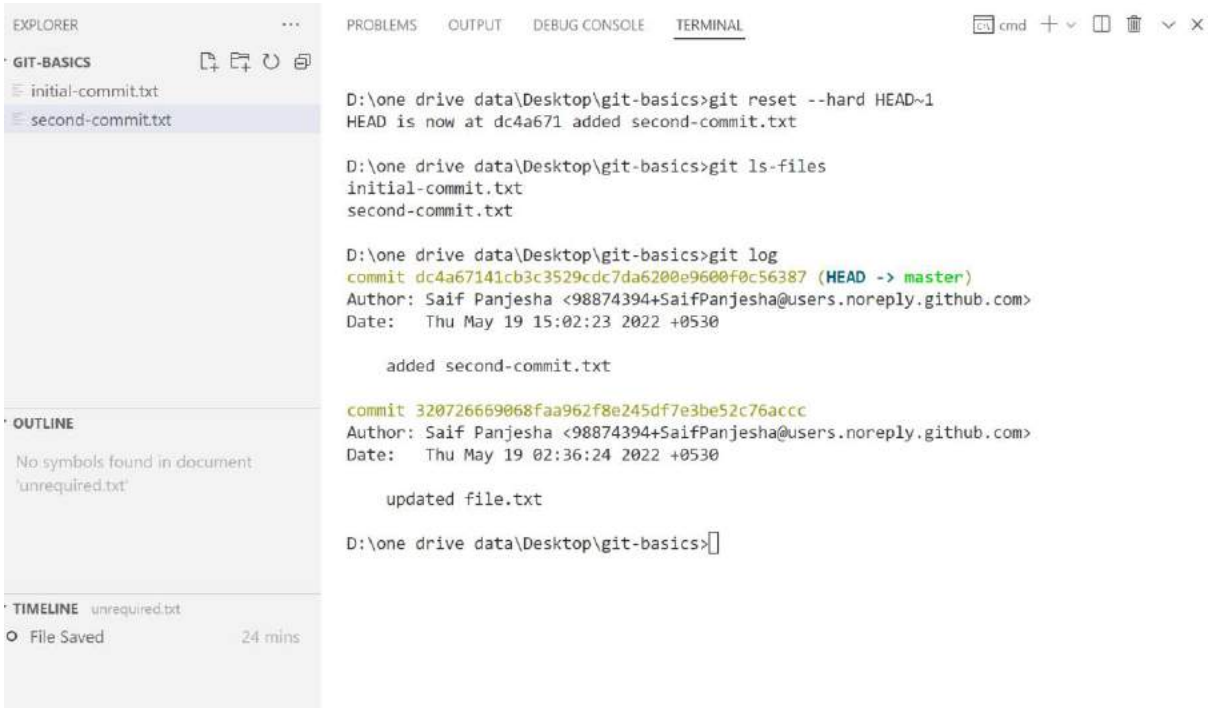


Fig. Head -> master to added second-commit.txt

Deleting Branches:

git branch -D <<branch-name>>:

-d: allows you to only delete the branches.

-D: allows you to force full delete your merge branches that not needed anymore

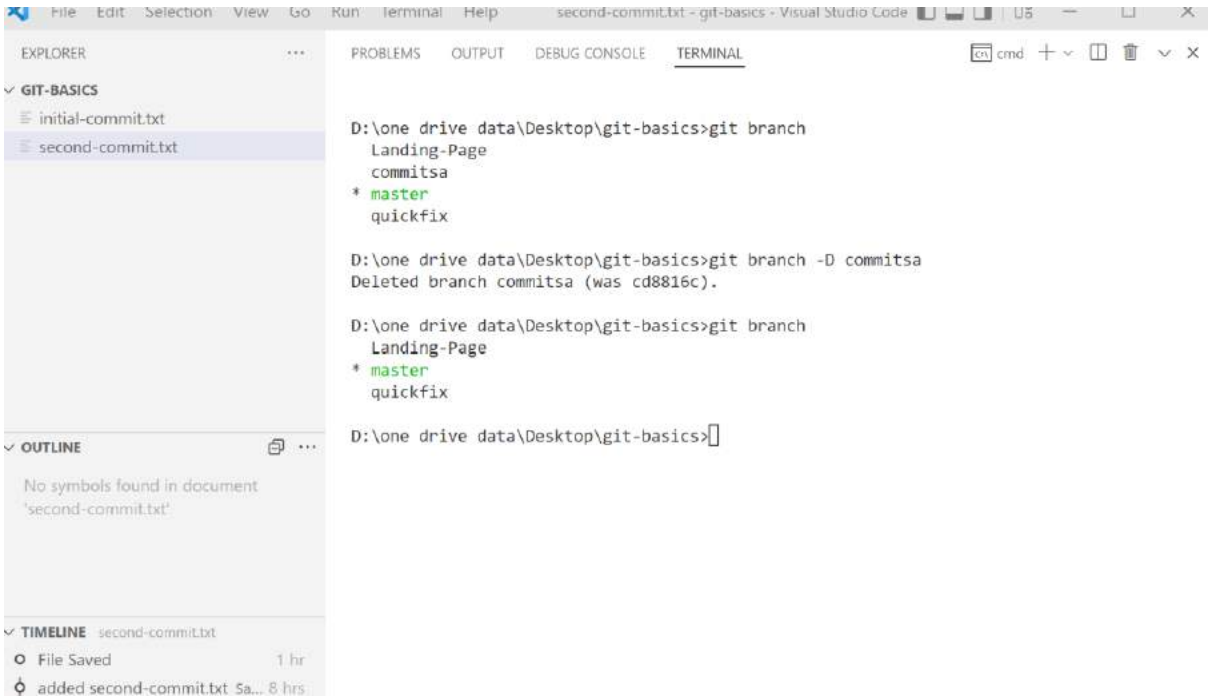


Fig. Commitsa Branch Deleted

To delete multiple branches:

```
D:\one drive data\Desktop\git-basics>git branch -D Landing-Page quickfix
Deleted branch Landing-Page (was cd8816c).
Deleted branch quickfix (was cd8816c).

D:\one drive data\Desktop\git-basics>
```

Fig. Deleted multiple branches

Committing Detached Head Changes:

The Commit which is not part of any branches is called detached head state.

```
D:\one drive data\Desktop\git-basics>git branch
* master

D:\one drive data\Desktop\git-basics>git log
commit dc4a67141cb3c3529cdc7da6200e9600f0c56387 (HEAD -> master)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Thu May 19 15:02:23 2022 +0530

    added second-commit.txt

commit 320726669068faa962f8e245df7e3be52c76accc
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Thu May 19 02:36:24 2022 +0530

    updated file.txt

D:\one drive data\Desktop\git-basics>
```

Fig. Master branch with two commits.

Creating a new File: dummy.txt

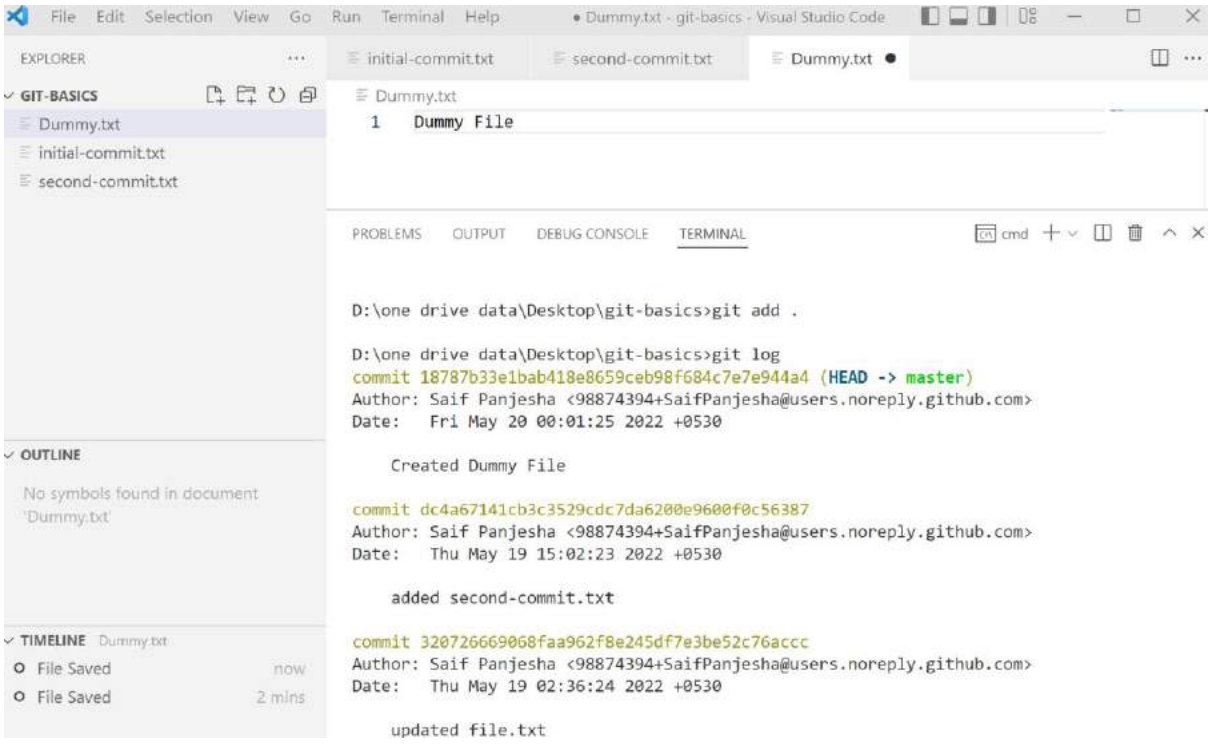


Fig. Dummy File

Detached State:

D:\one drive data\Desktop\git-basics>git checkout 320726669068faa962f8e245df7e3be52c76accc

Note: switching to '320726669068faa962f8e245df7e3be52c76accc'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 3207266 updated file.txt

D:\one drive data\Desktop\git-basics>git log
commit 320726669068faa962f8e245df7e3be52c76accc (HEAD)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date: Thu May 19 02:36:24 2022 +0530

updated file.txt

D:\one drive data\Desktop\git-basics>git branch
*** (HEAD detached at 3207266)**
master

Got Unstaged or Untracked File

```
D:\one drive data\Desktop\git-basics>git status
HEAD detached at 3207266
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   initial-commit.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        detached-head.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

```
D:\one drive data\Desktop\git-basics>git add .

D:\one drive data\Desktop\git-basics>git status

HEAD detached at 3207266
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   initial-commit.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        detached-head.txt

no changes added to commit (use "git add" and/or "git commit -a")

D:\one drive data\Desktop\git-basics>git add .

D:\one drive data\Desktop\git-basics>git status
HEAD detached at 3207266
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   detached-head.txt
        modified:   initial-commit.txt
```

Let's Commit Now:

```
D:\one drive data\Desktop\git-basics>git commit -m "Changes in detached head"
[detached HEAD b44e221] Changes in detached head
 2 files changed, 1 insertion(+)
 create mode 100644 detached-head.txt

D:\one drive data\Desktop\git-basics>git log
commit b44e221714e698dbb0f2d83643204078455461l2 (HEAD)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Fri May 20 00:18:55 2022 +0530

    Changes in detached head

commit 320726669068faa962f8e245df7e3be52c76accc
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Thu May 19 02:36:24 2022 +0530

    updated file.txt

D:\one drive data\Desktop\git-basics>git branch
* (HEAD detached from 3207266)
  master
 ----------------------------------------------------------------------------------------------------

D:\one drive data\Desktop\git-basics>git switch master
Warning: you are leaving 1 commit behind, not connected to
any of your branches:
  b44e221 Changes in detached head
If you want to keep it by creating a new branch, this may be a good time
to do so with:
 git branch <new-branch-name> b44e221
Switched to branch 'master'
```

Detached branch gone:

```
D:\one drive data\Desktop\git-basics>git branch
* master
```

```
D:\one drive data\Desktop\git-basics>git branch detached-head  b44e221

D:\one drive data\Desktop\git-basics>git branch
  detached-head
* master

D:\one drive data\Desktop\git-basics>git checkout detached-head
Switched to branch 'detached-head'

D:\one drive data\Desktop\git-basics>git switch master
Switched to branch 'master'

D:\one drive data\Desktop\git-basics>git merge detached-head
Merge made by the 'ort' strategy.
 detached-head.txt  | 0
 initial-commit.txt | 1 +
 2 files changed, 1 insertion(+)
 create mode 100644 detached-head.txt
```

Succesful Committing Detached Head Changes to master:

```
D:\one drive data\Desktop\git-basics>git ls-files
detached-head.txt
dummy.txt
initial-commit.txt
second-commit.txt
```

```
D:\one drive data\Desktop\git-basics>git log
commit bd822cb7c7880499bafe6d035c08e6142bb9aa19 (HEAD -> master)
Merge: 18787b3 b44e221
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Fri May 20 00:30:11 2022 +0530

    Merge branch 'detached-head'
```

commit b44e221714e698dbb0f2d83643204078455461l2 (detached-head)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date: Fri May 20 00:18:55 2022 +0530

    Changes in detached head

commit 18787b33e1bab418e8659ceb98f684c7e7e944a4
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date: Fri May 20 00:01:25 2022 +0530

    Created Dummy File

commit dc4a67141cb3c3529cdc7da6200e9600f0c56387
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date: Thu May 19 15:02:23 2022 +0530

    added second-commit.txt

commit 320726669068faa962f8e245df7e3be52c76accc
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date: Thu May 19 02:36:24 2022 +0530

    updated file.txt

D:\one drive data\Desktop\git-basics>git checkout
dc4a67141cb3c3529cdc7da6200e9600f0c56387
Note: switching to 'dc4a67141cb3c3529cdc7da6200e9600f0c56387'.

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at dc4a671 added second-commit.txt

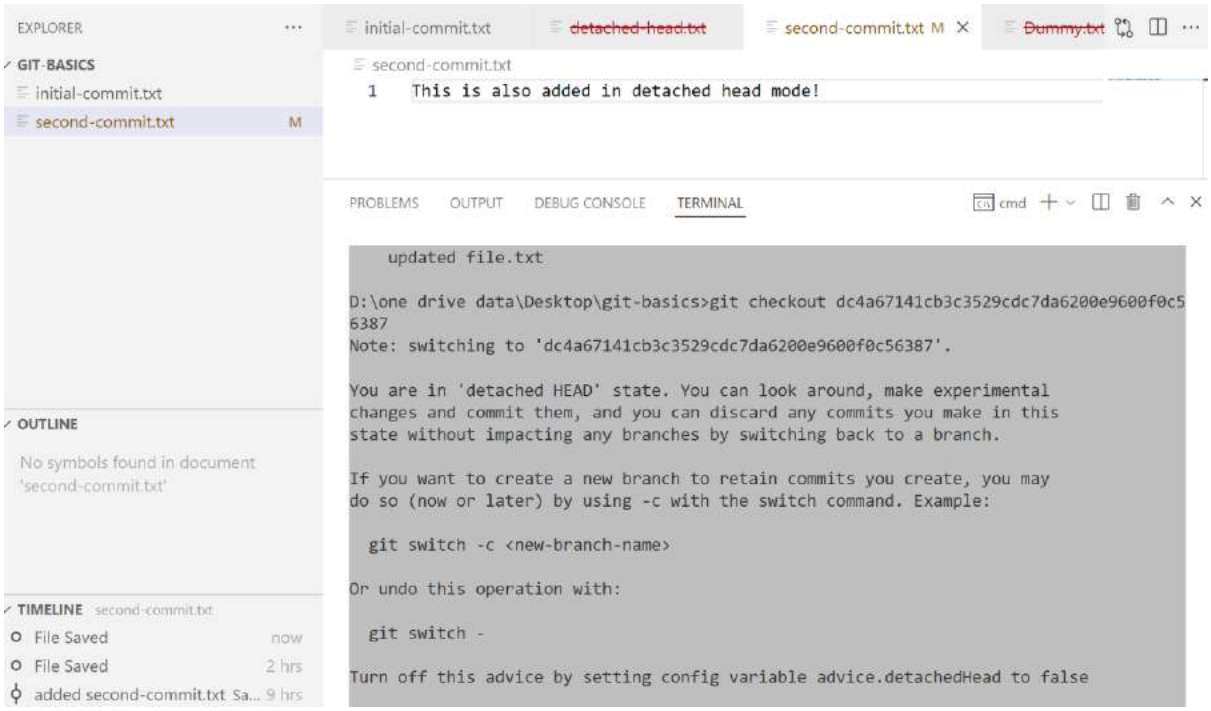


Fig. Added Some in Detached Head

```
D:\one drive data\Desktop\git-basics>git branch
* (HEAD detached at dc4a671)
  detached-head
  master

D:\one drive data\Desktop\git-basics>git add .

D:\one drive data\Desktop\git-basics>git commit -m "Text added to Detached Head"
[detached HEAD 4578010] Text added to Detached Head
 1 file changed, 1 insertion(+)

D:\one drive data\Desktop\git-basics>
```

Fig. Committed Save Changes

Successful Committing Detached Head Changes to master:

D:\one drive data\Desktop\git-basics>git branch Detached2Head
D:\one drive data\Desktop\git-basics>git branch
* (HEAD detached from dc4a671)
  Detached2Head
  detached-head
  master
D:\one drive data\Desktop\git-basics>git switch master
Previous HEAD position was 4578010 Text added to Detached Head
Switched to branch 'master'

D:\one drive data\Desktop\git-basics>git merge Detached2Head
Merge made by the 'ort' strategy.
 second-commit.txt | 1 +
 1 file changed, 1 insertion(+)

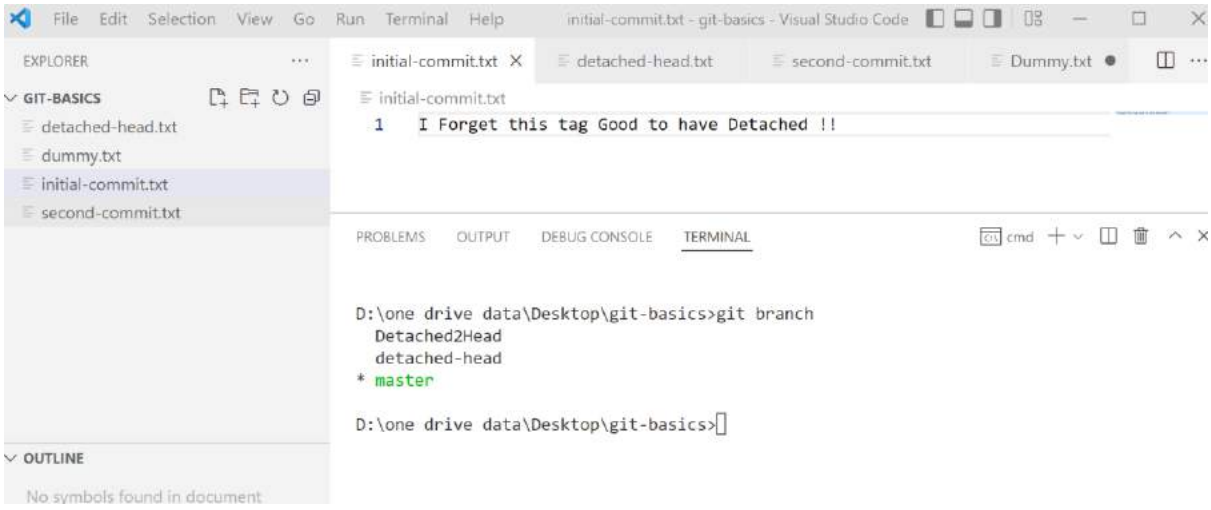


Fig. Succesful head Changes

D:\one drive data\Desktop\git-basics>git branch -D Detached2Head detached-head
Deleted branch Detached2Head (was 4578010).
Deleted branch detached-head (was b44e221).

D:\one drive data\Desktop\git-basics>git branch
* master

Understanding. gitignore:

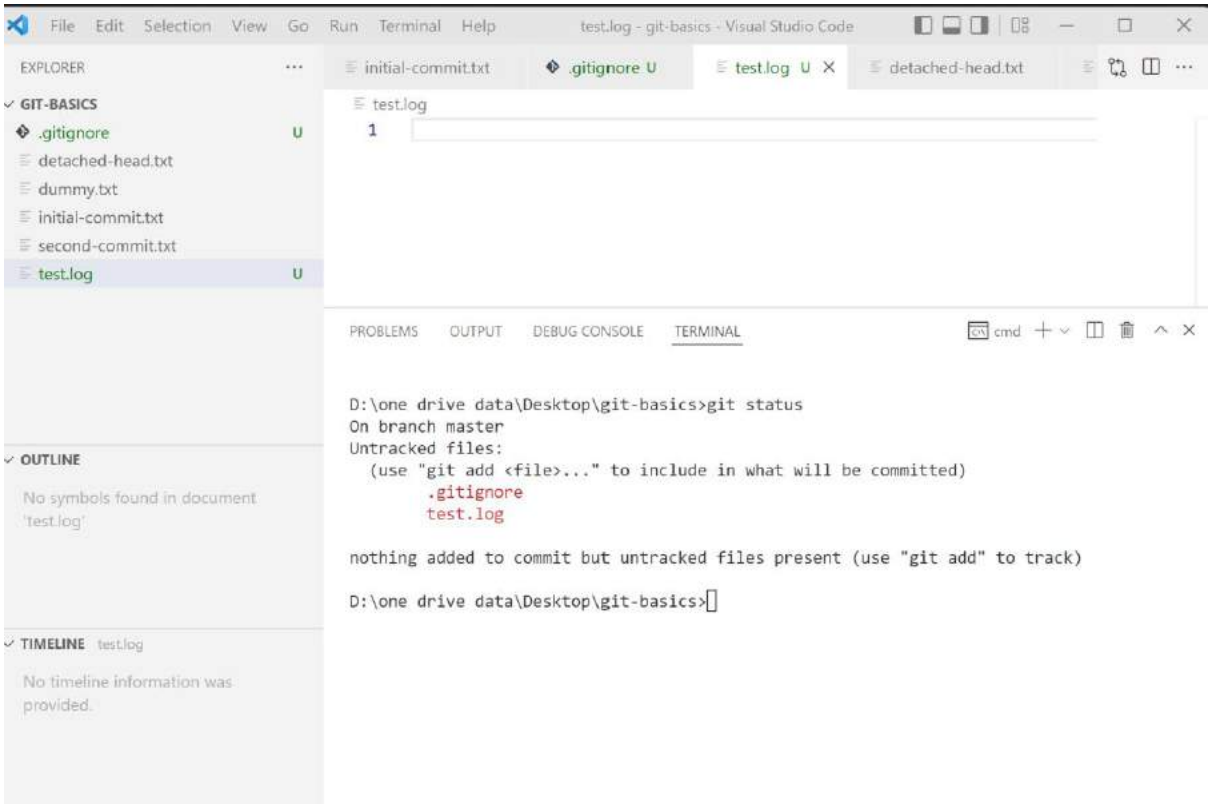


Fig. ignore file created

Ignoring test.log File

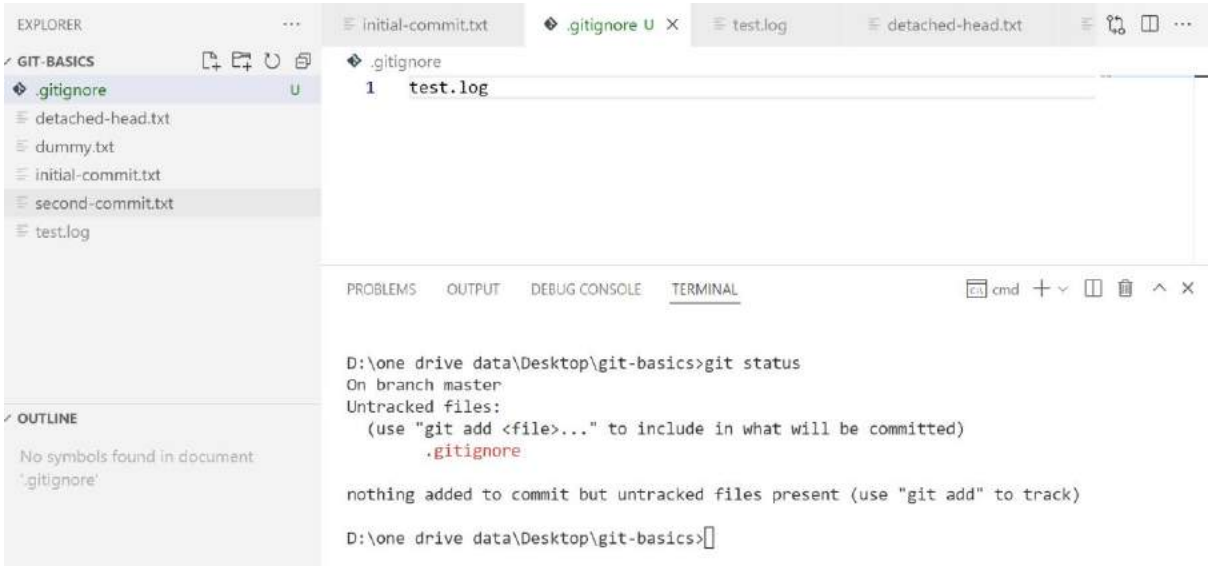


Fig. test.log File ignored

To Ignore Multiple Files:

***.log – for ignorance all files**

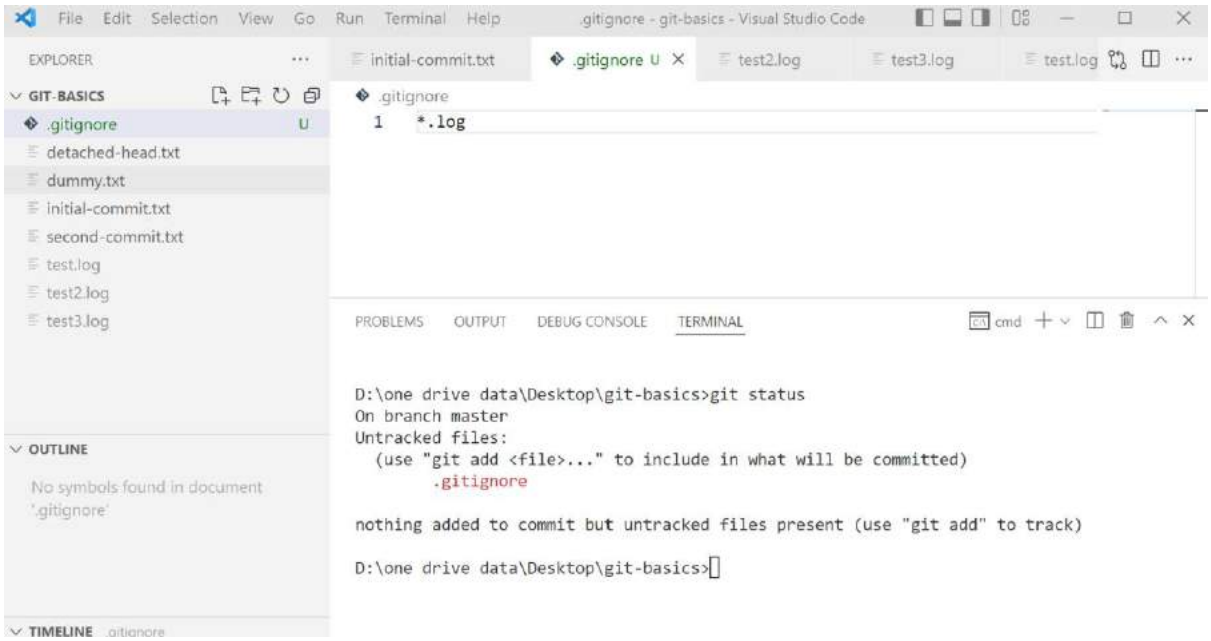


Fig. Multiple Fie get ignored test, test2 and test3

Not to Ignore files "!test.log" – not ignoring the files

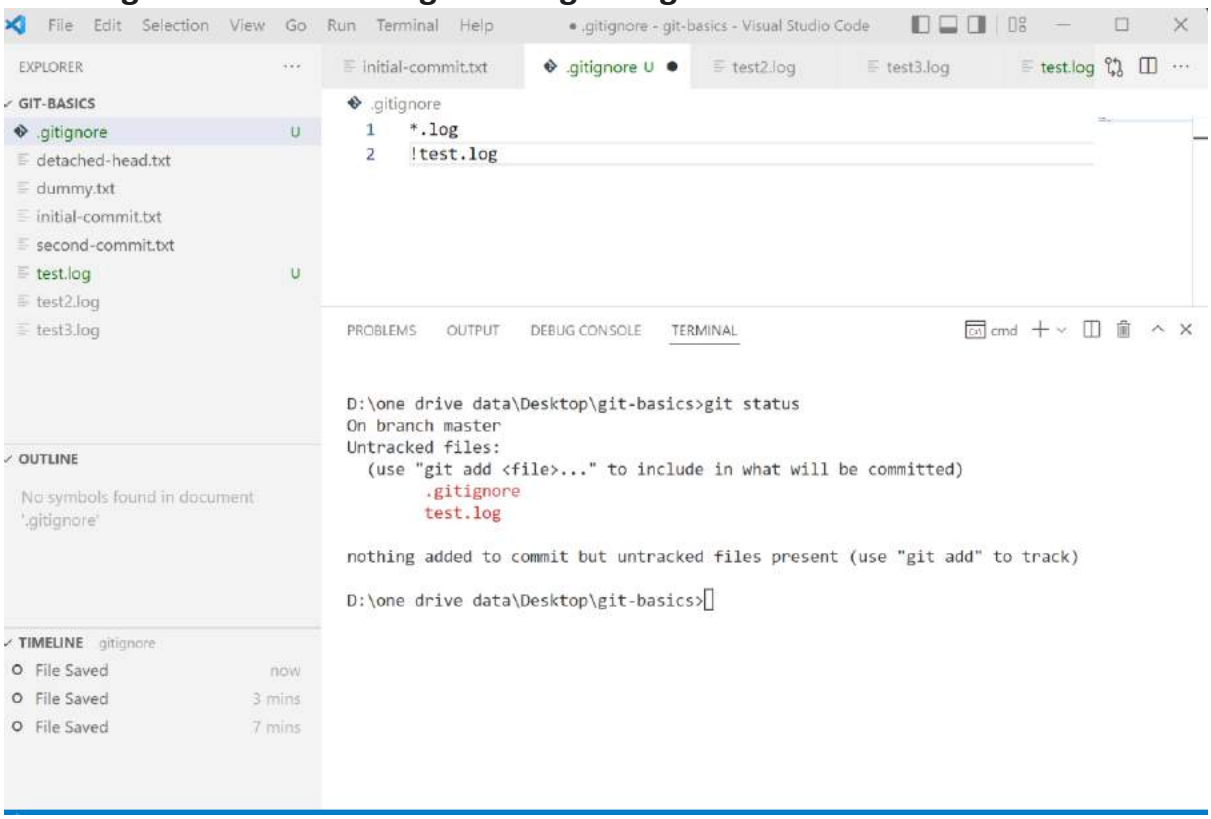


Fig.Not ignored

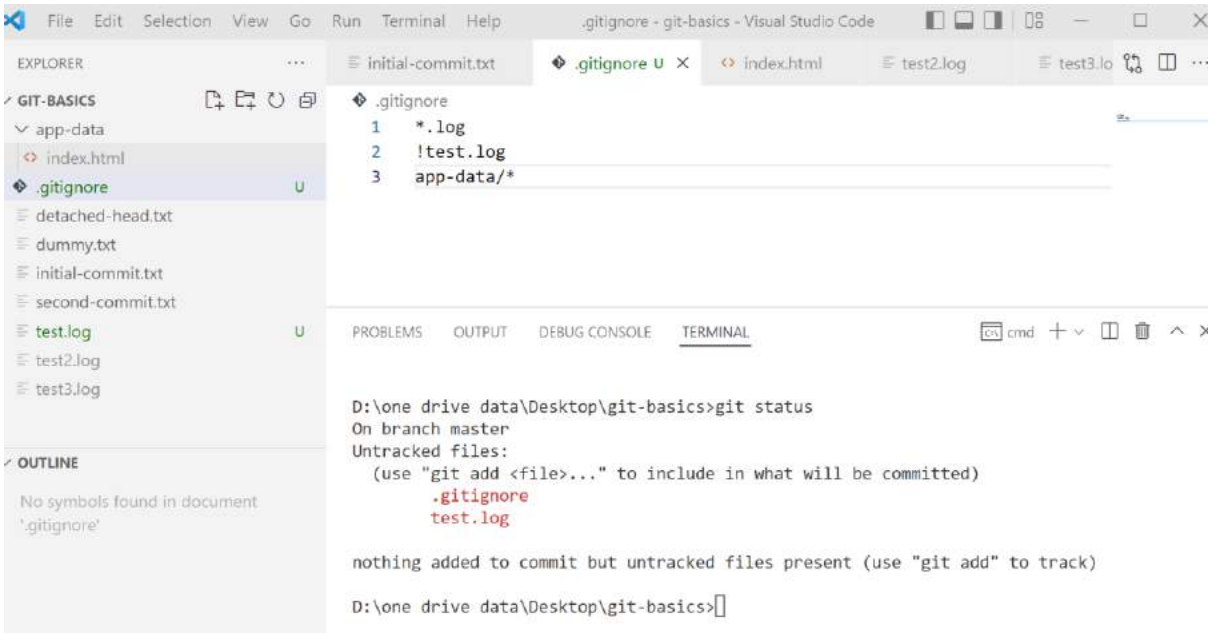


Fig. ignored folders “/ “command

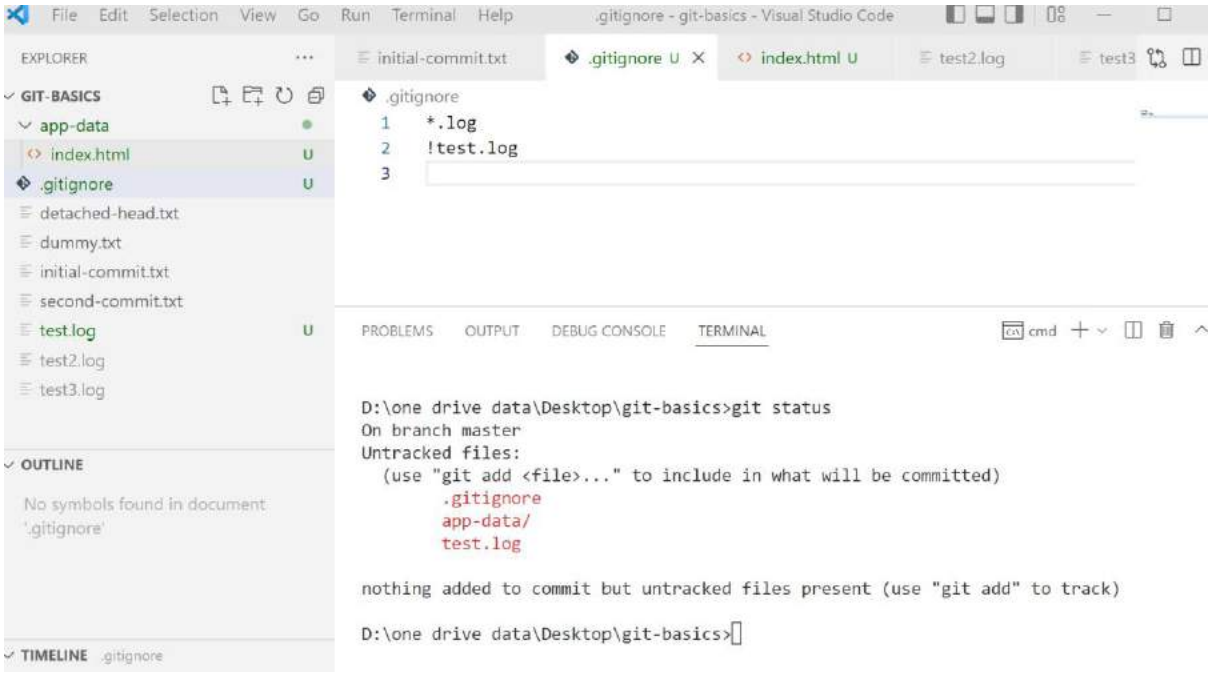


Fig. Not ignored folders without “/ “command

Clean all Files & Folders: **Deletes untracked Files**

D:\one drive data\Desktop\git-basics>git clean -df

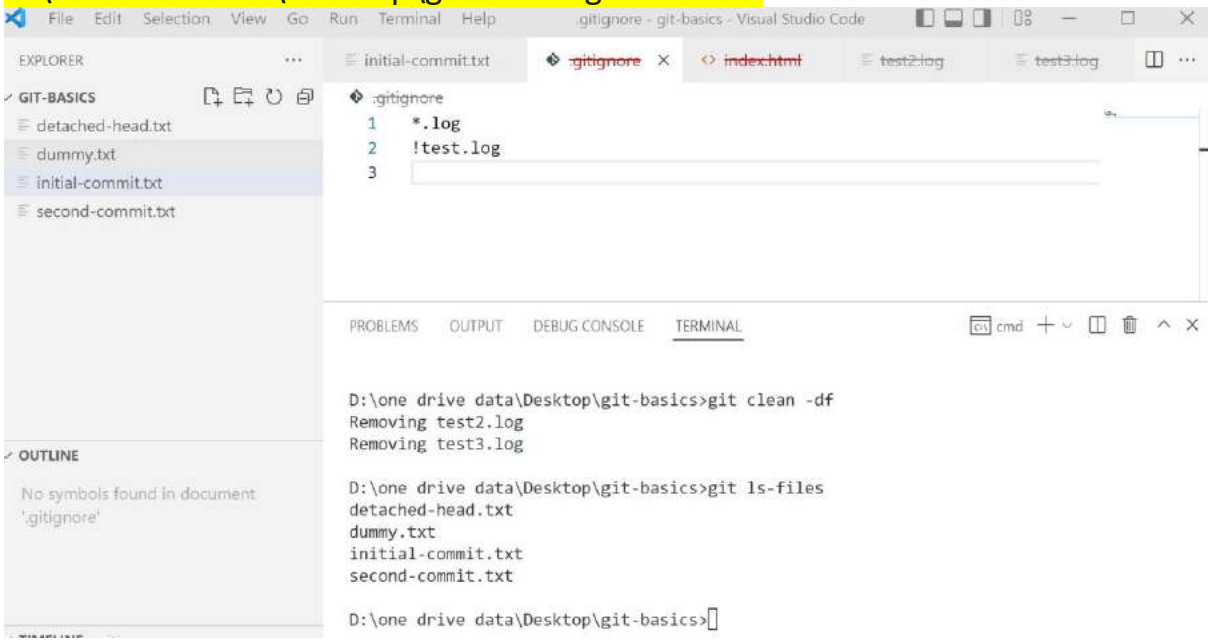


Fig. Clean all files & folders

Wrap Up What We Learn:

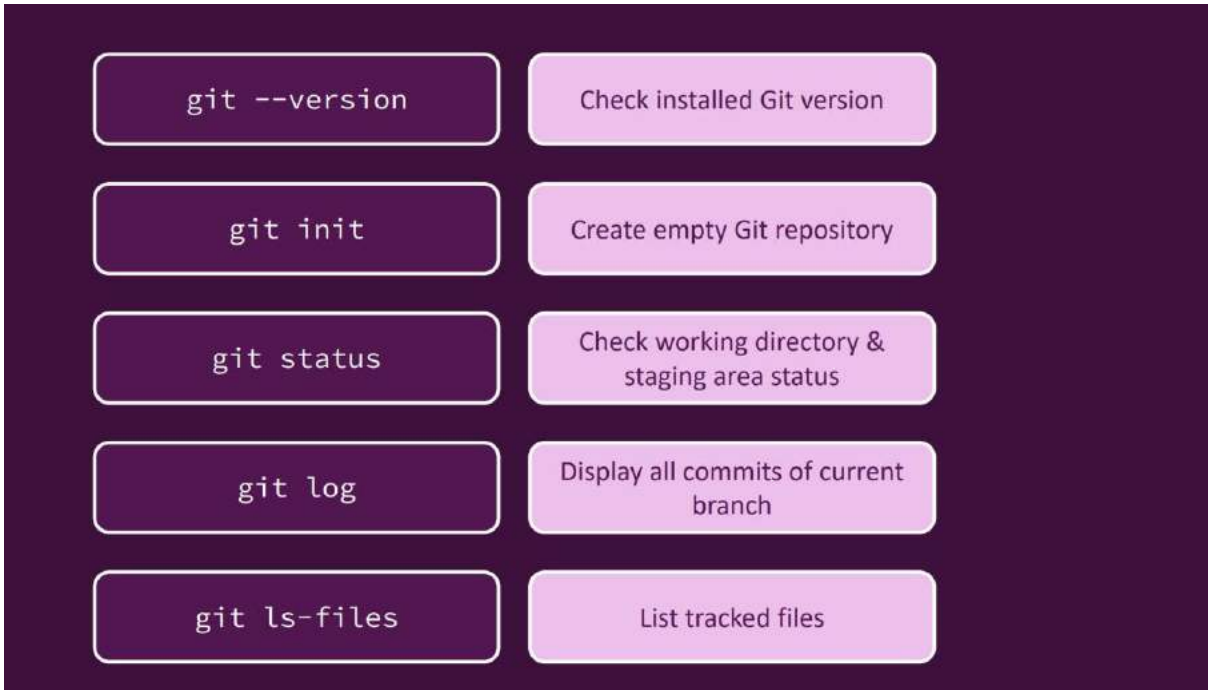


Fig . General Commands

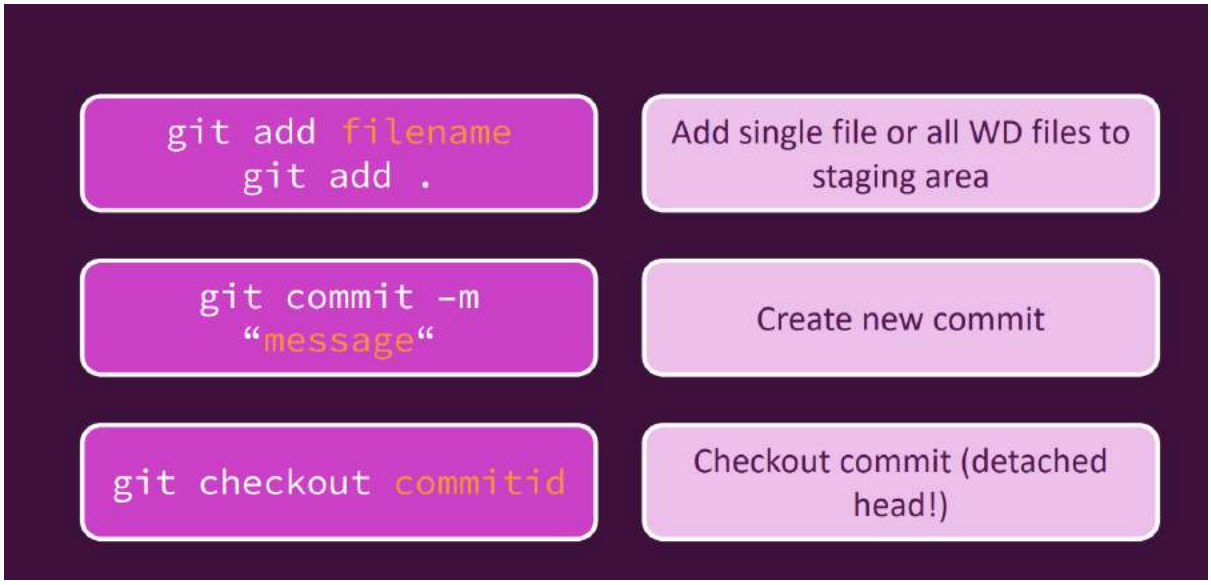


Fig. Basic Commit Creation and access

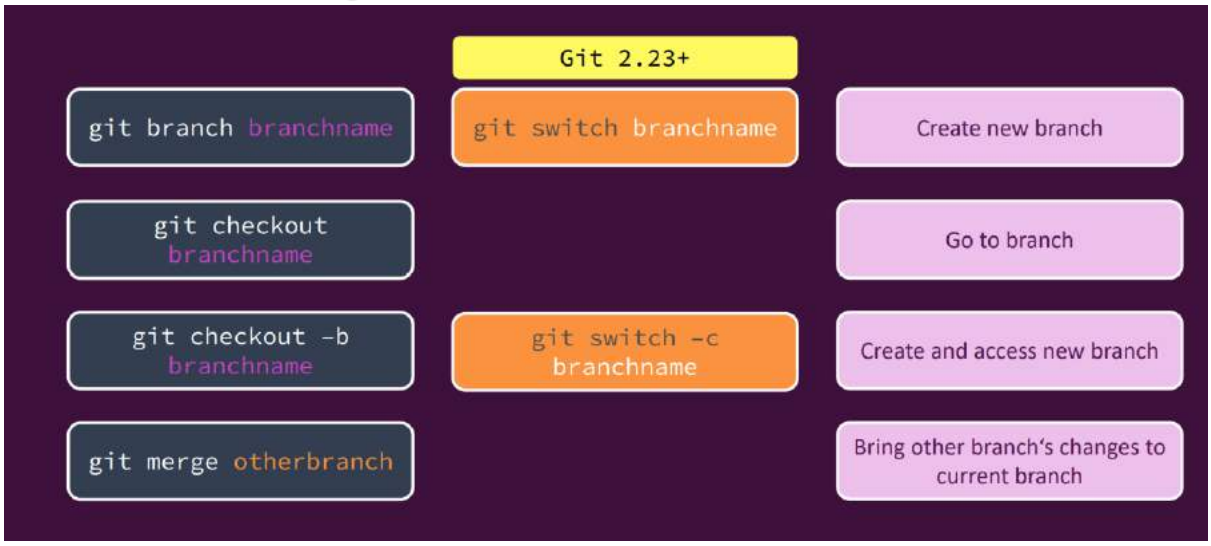


Fig. Branch and creation access

Fig. Deleting Data Summary

Diving Deep into Git

Fig. Module Introduction

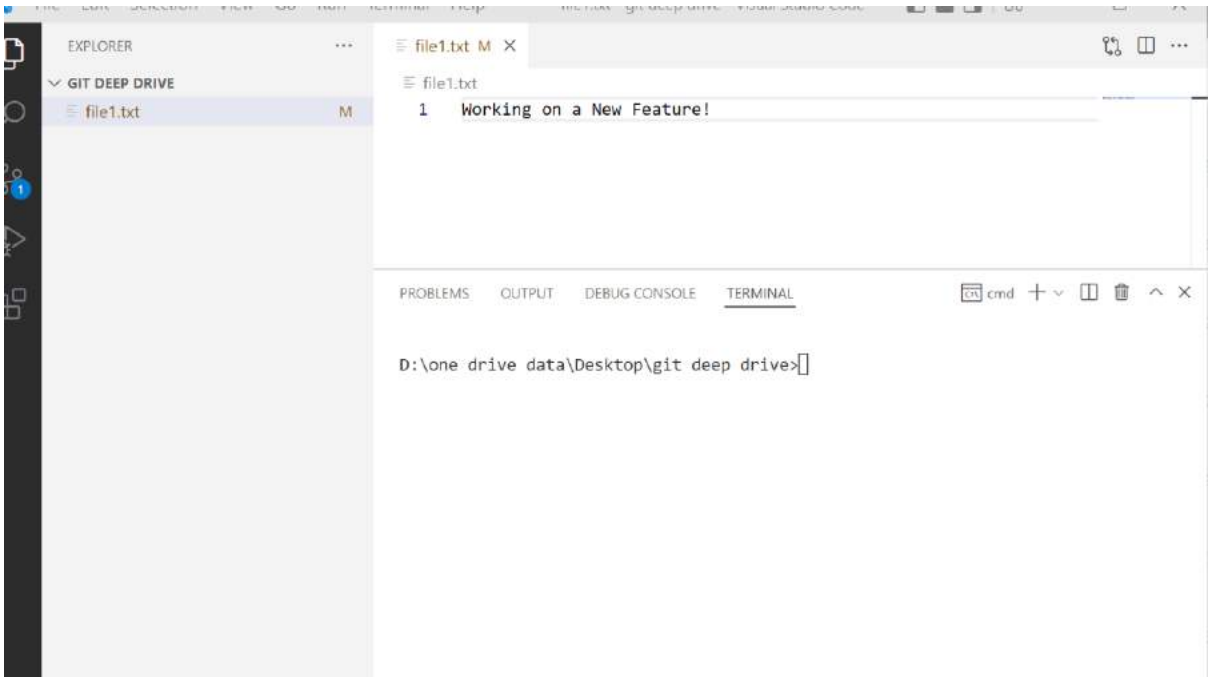


Fig. Created new file into Git Deep Drive Folder

Understanding the Stash ("git stash"): The stash is kind of an internal memory where you can save uncommitted Unstaged changes**.**

git stash: temporarily shelves (or stashes) change you've made to your working copy so you can work on something else, and then come back and re-apply them later on.

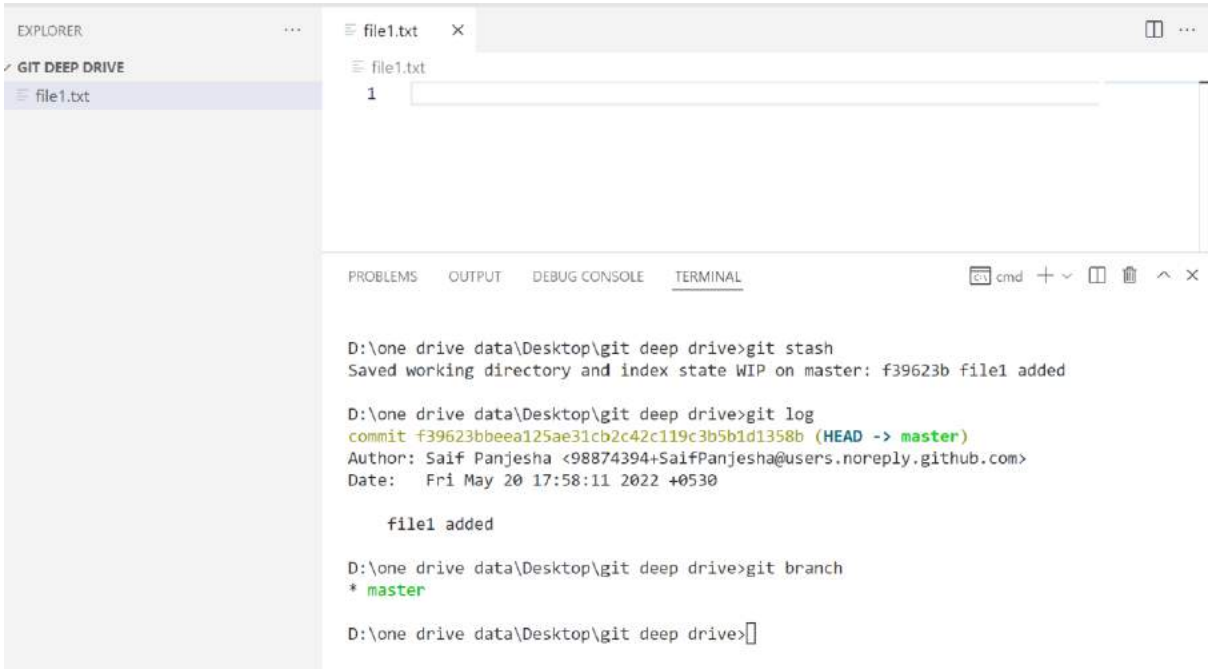


Fig. git stash or shelves changes

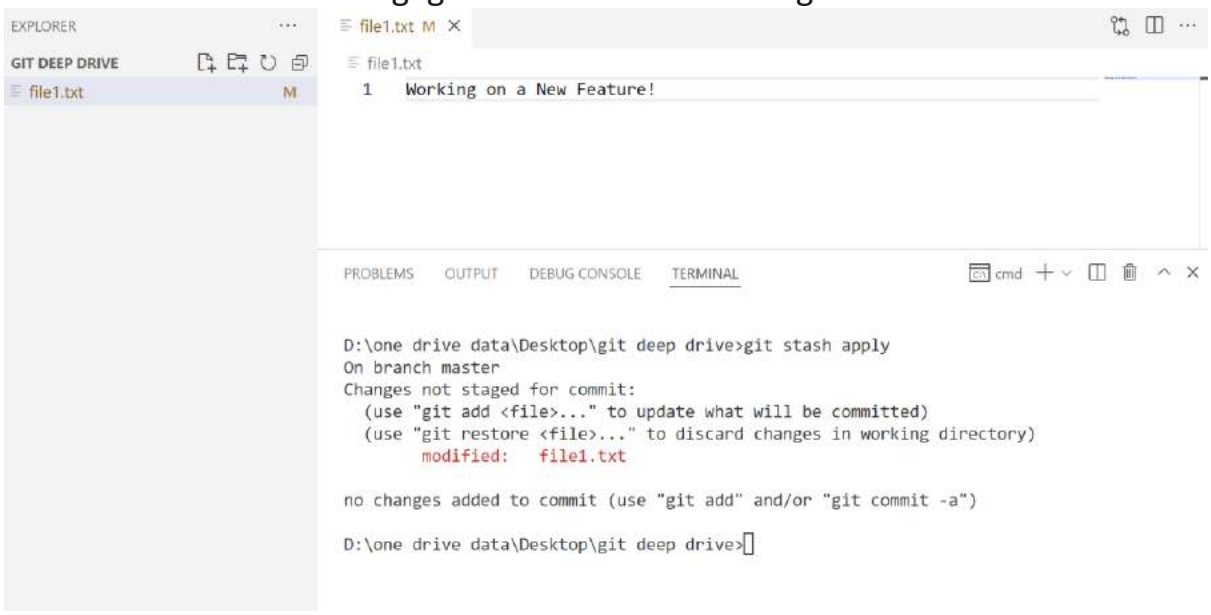


Fig. git stash apply

Continue Working With Another Awesome Feature In Code .

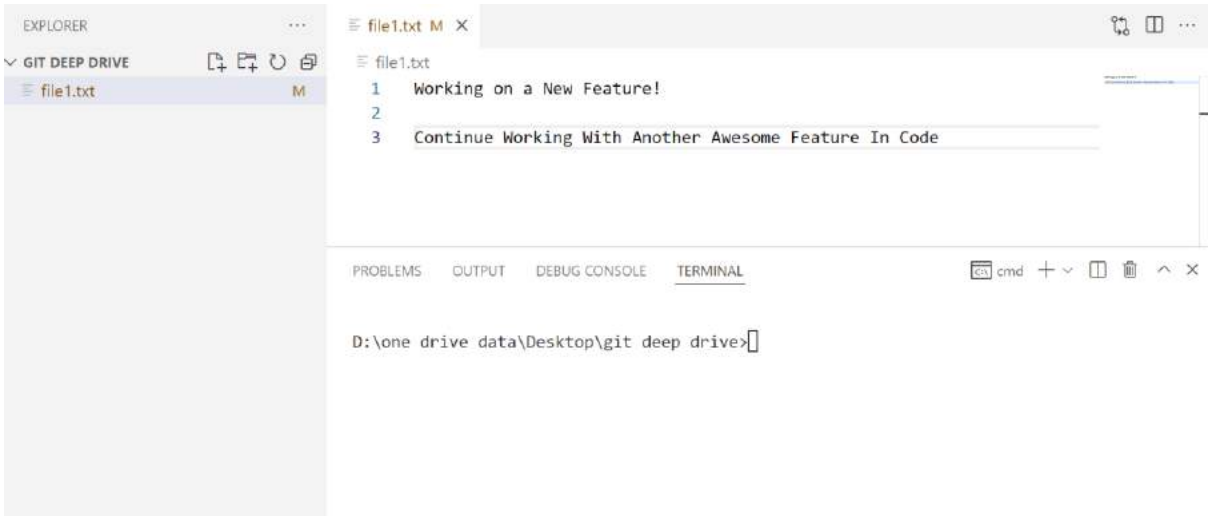


Fig. Added New Feature in Code

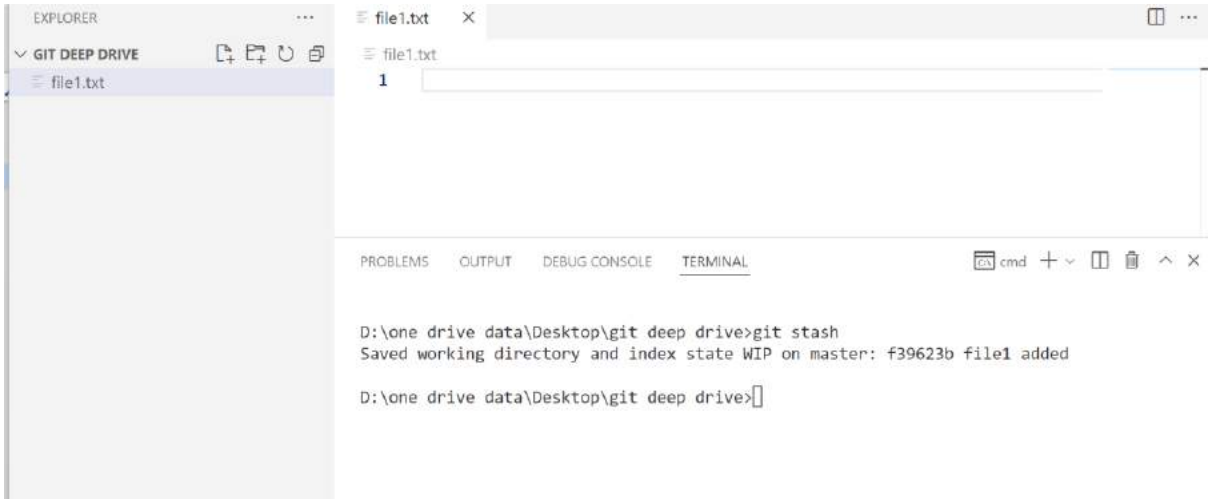


Fig. git stash Work Saved

Fig. git stash apply

git stash list: List of all Stash Changes

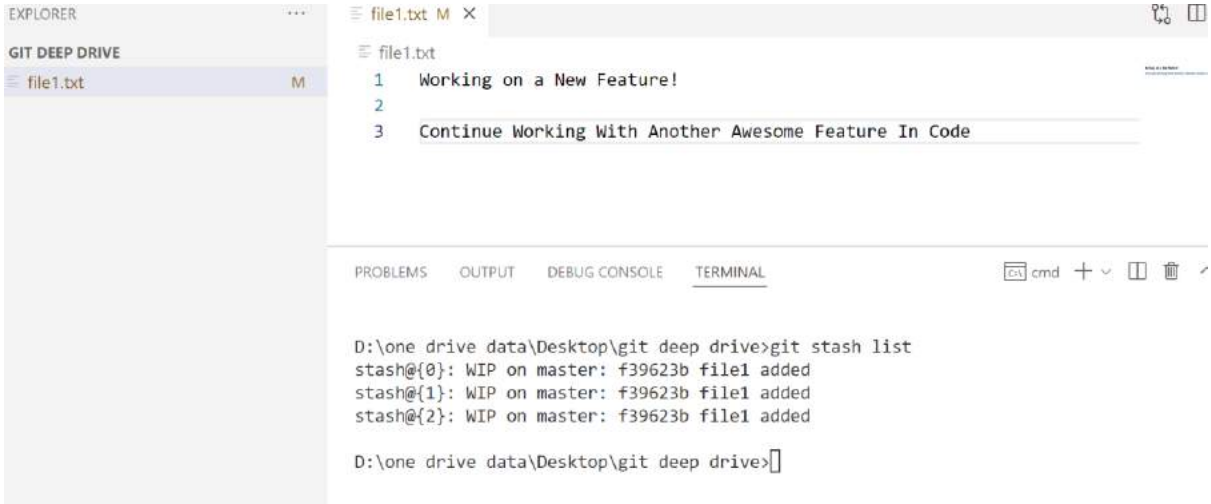


Fig. git stash list

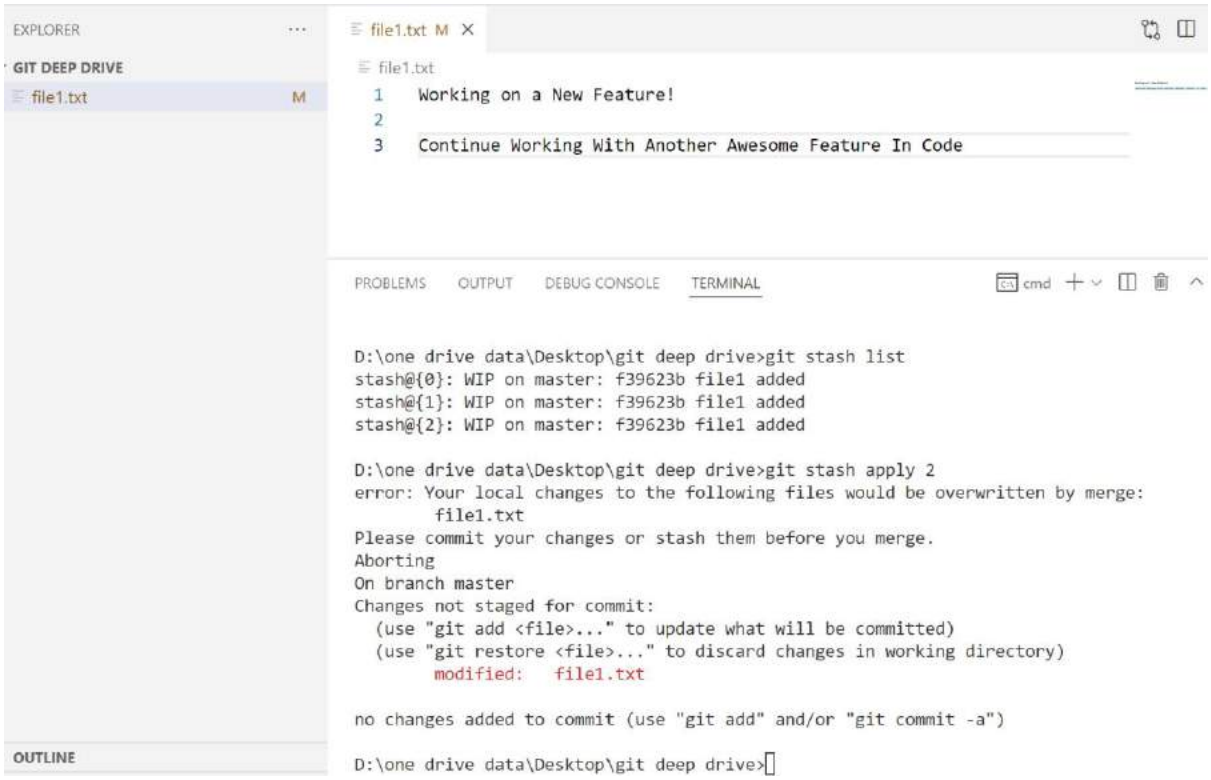


Fig. Error Changes not saved we need to commit, add or stash the changes

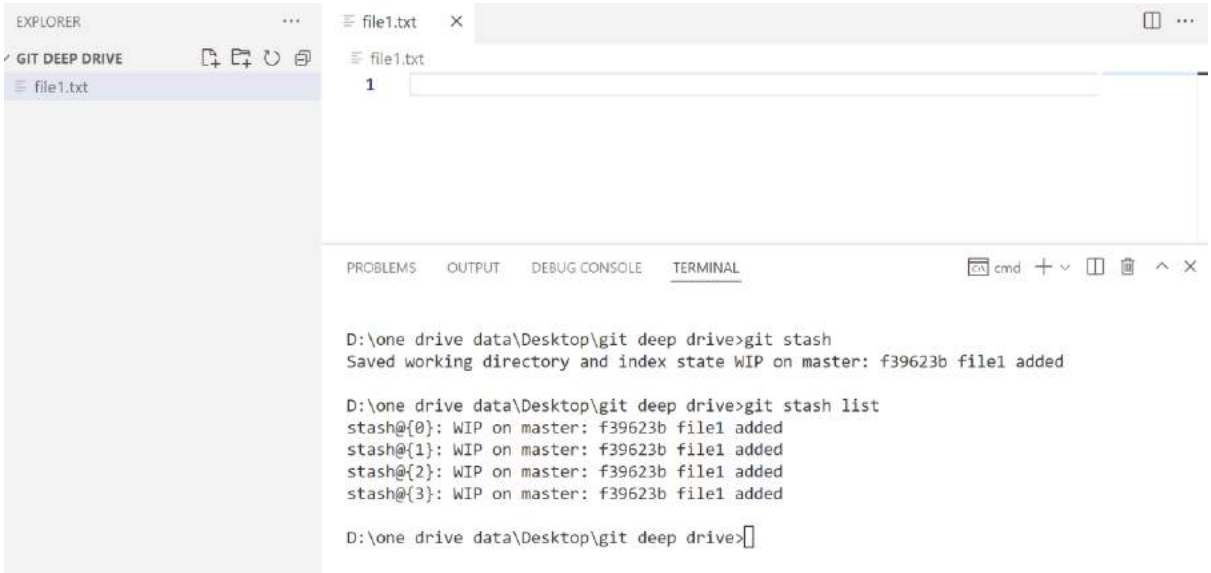


Fig. git stash changes saved in working directory

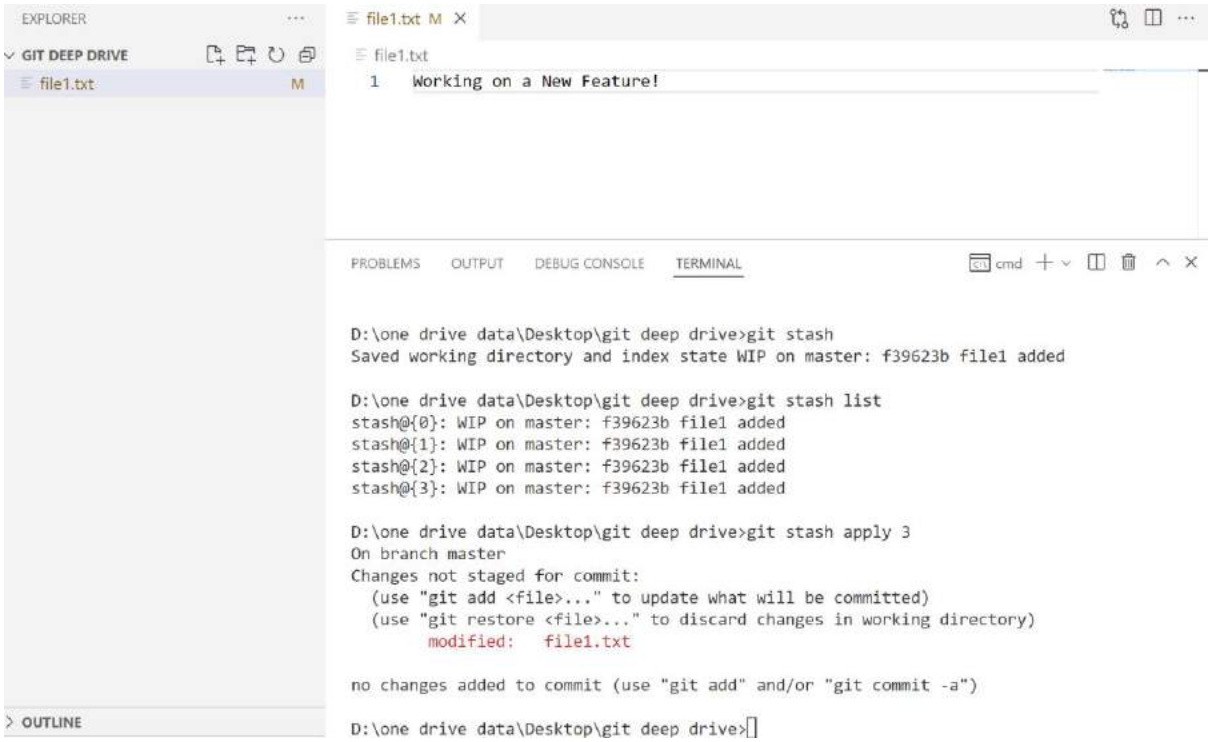


Fig. git stash apply 2 shows we are working on new feature

To check our latest changes:

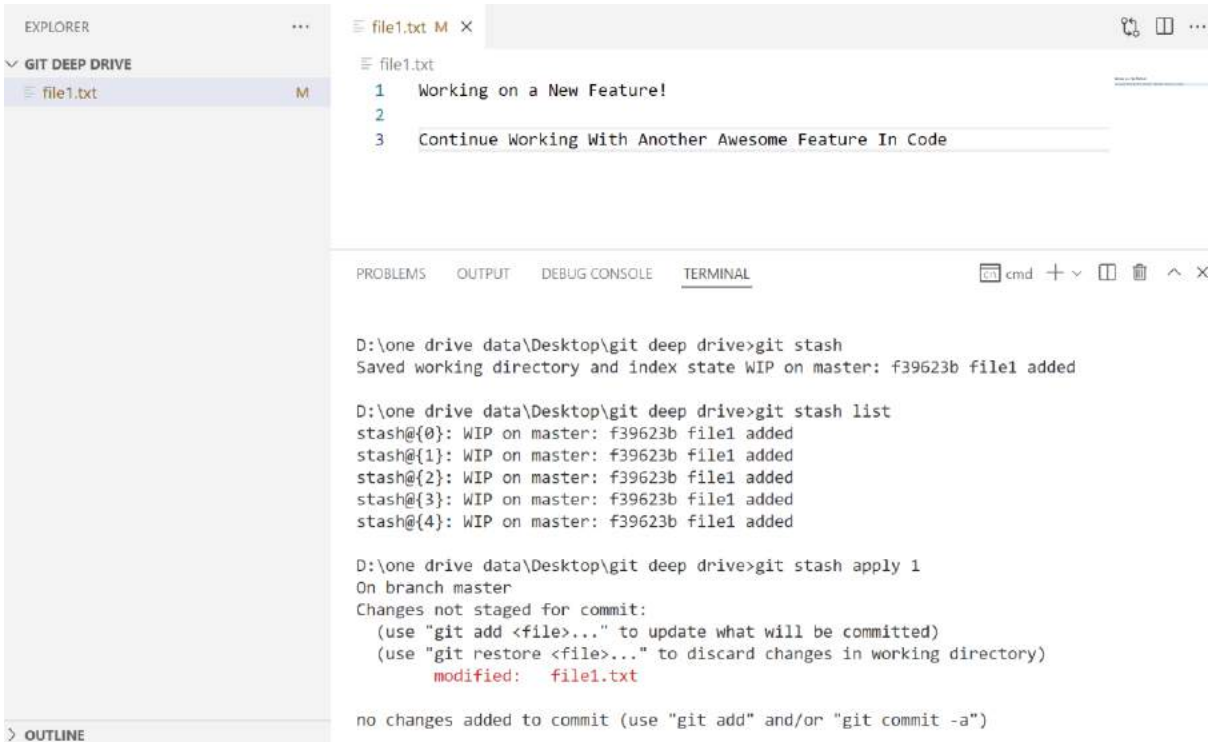


Fig. To Check our Latest changes with git stash apply

We can add message to our stash:

Fig. Third Feature added

stash push -m "Third Feature added"

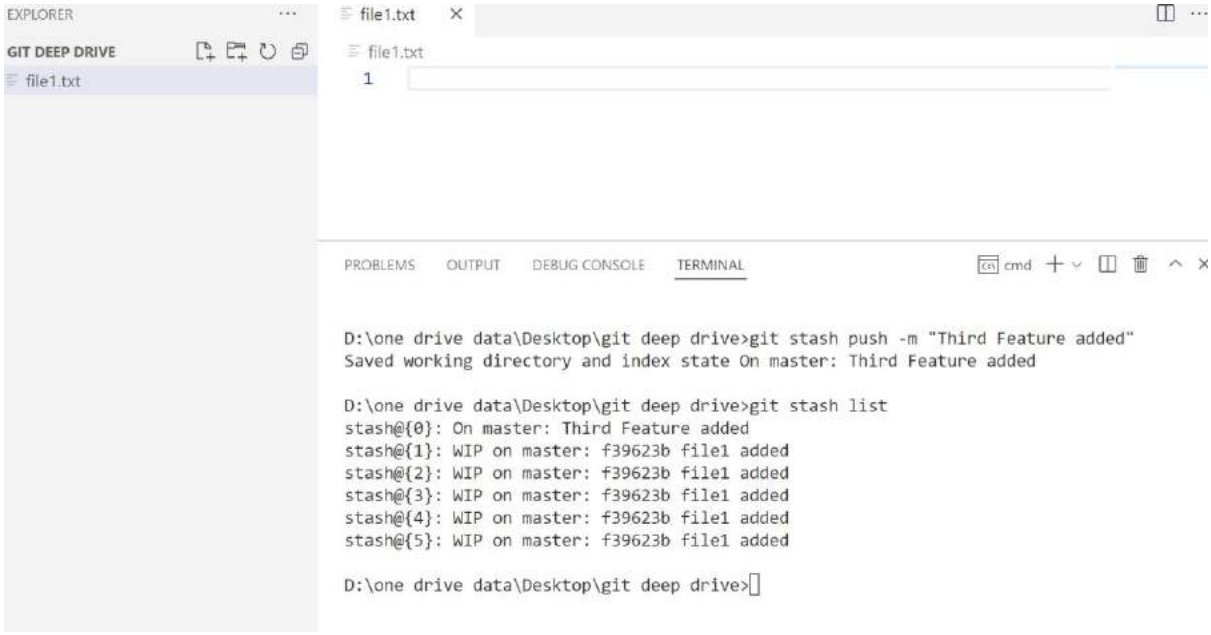
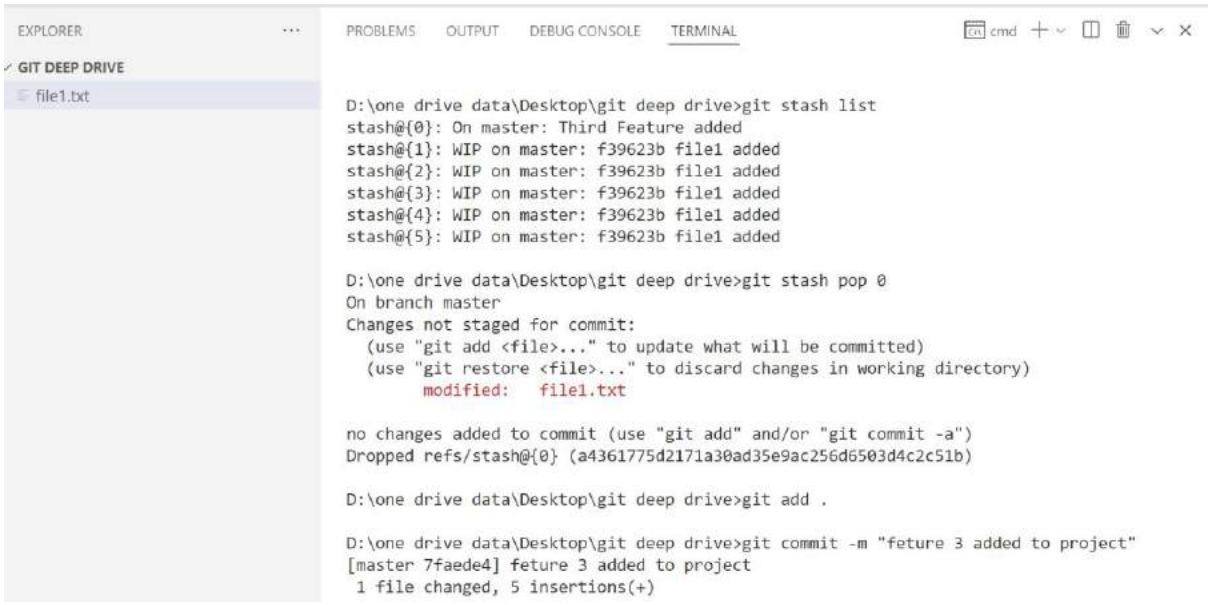


Fig. shows we can identify different feature in our stash with the Message

Now this message we have to add in our Projects:




```
D:\one drive data\Desktop\git deep drive>git stash list
stash@{0}: WIP on master: f39623b file1 added
stash@{1}: WIP on master: f39623b file1 added
stash@{2}: WIP on master: f39623b file1 added
stash@{3}: WIP on master: f39623b file1 added
stash@{4}: WIP on master: f39623b file1 added

D:\one drive data\Desktop\git deep drive>
```

```
D:\one drive data\Desktop\git deep drive>git log
commit 7faede4ff4b3e58cbb6580c10094065c7458367b (HEAD -> master)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Fri May 20 18:33:36 2022 +0530

    feture 3 added to project

commit f39623bbeea125ae31cb2c42c119c3b5b1d1358b
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Fri May 20 17:58:11 2022 +0530

    file1 added

D:\one drive data\Desktop\git deep drive>
```

Fig. Added Successful to project and out from stash stack

git stash clear: Remove the git stash

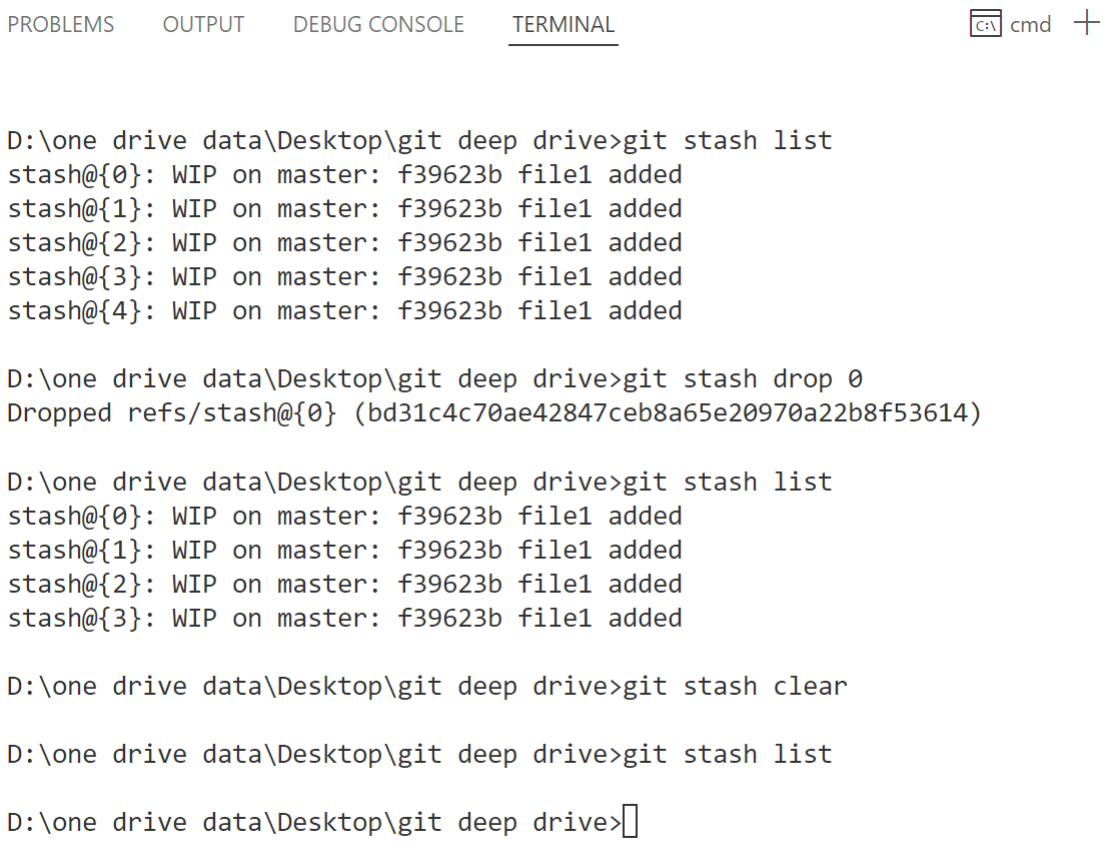


Fig. Remove the stash

Bringing the Lost data with “git reflog”

Let’s create a new file file2.txt and add in your project

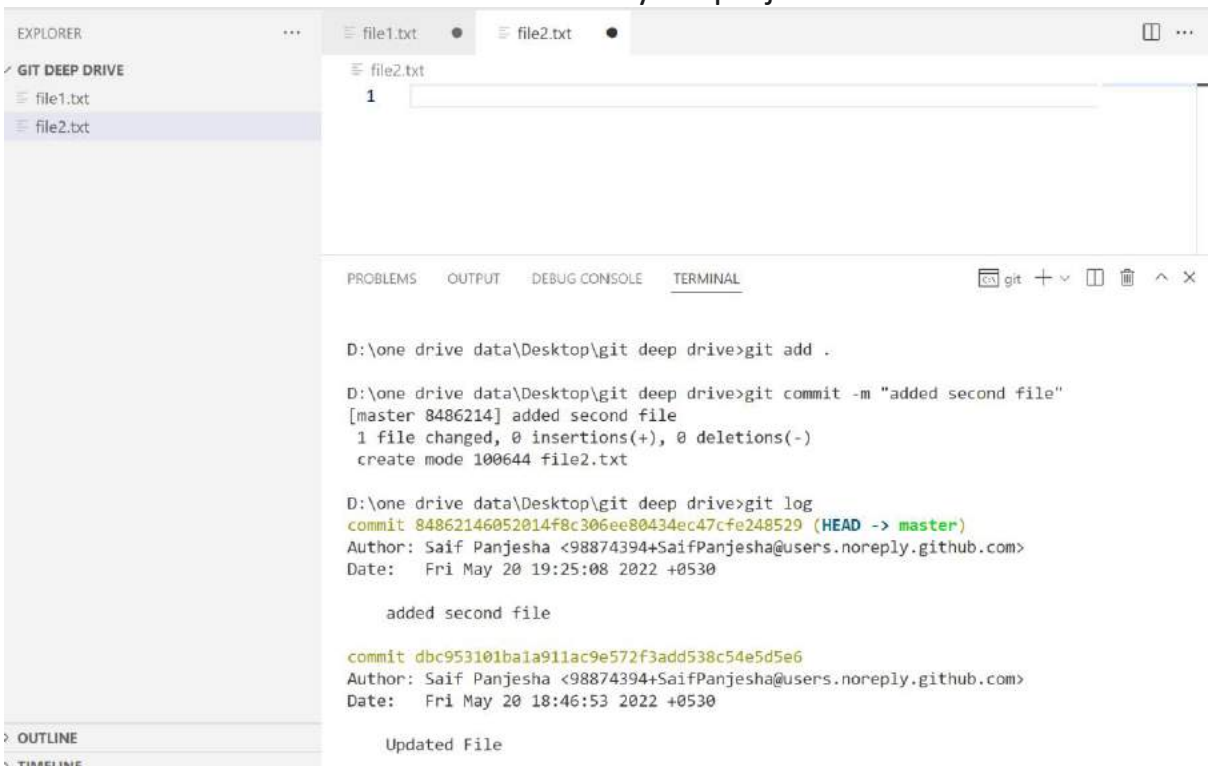


Fig. file2.txt created

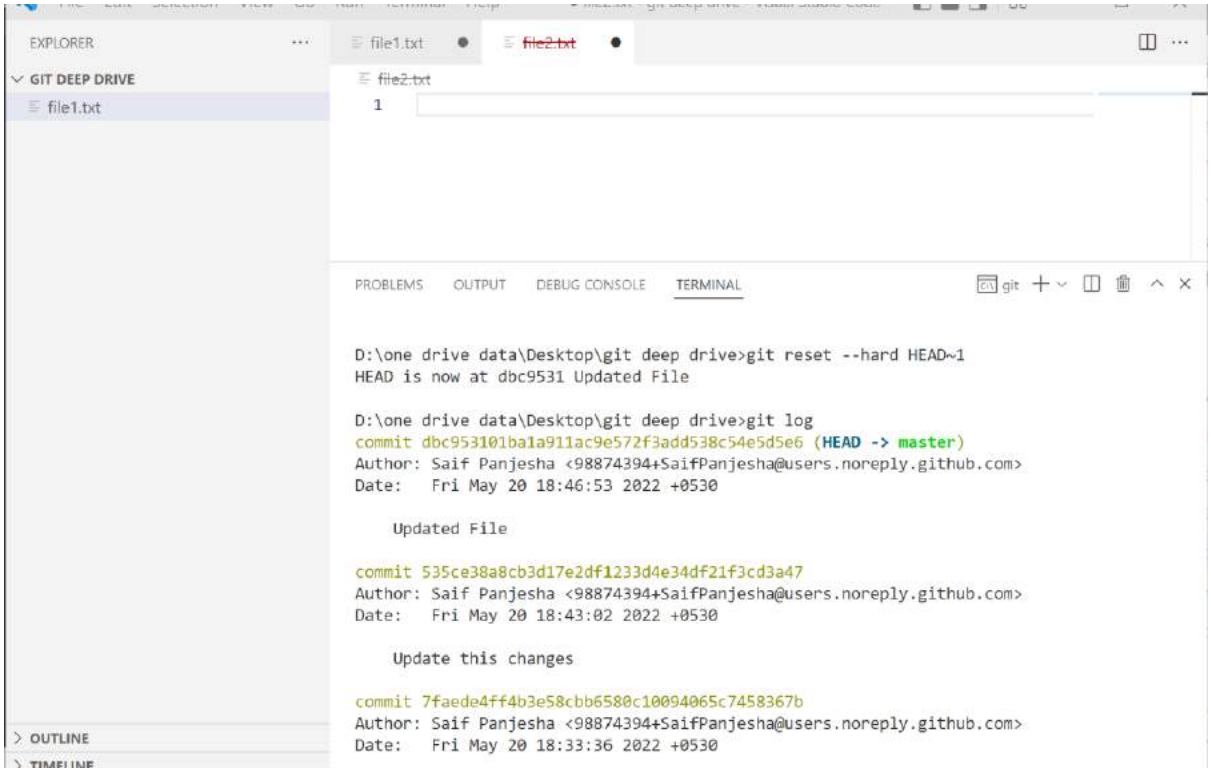


Fig. removed file not in used

Now to restore for next operations the file we need “git reflog”:
Git keeps track of updates to the tip of branches using a mechanism called reference logs, or "reflogs."

A branch tip is **the last commit or most recent commit on a branch**

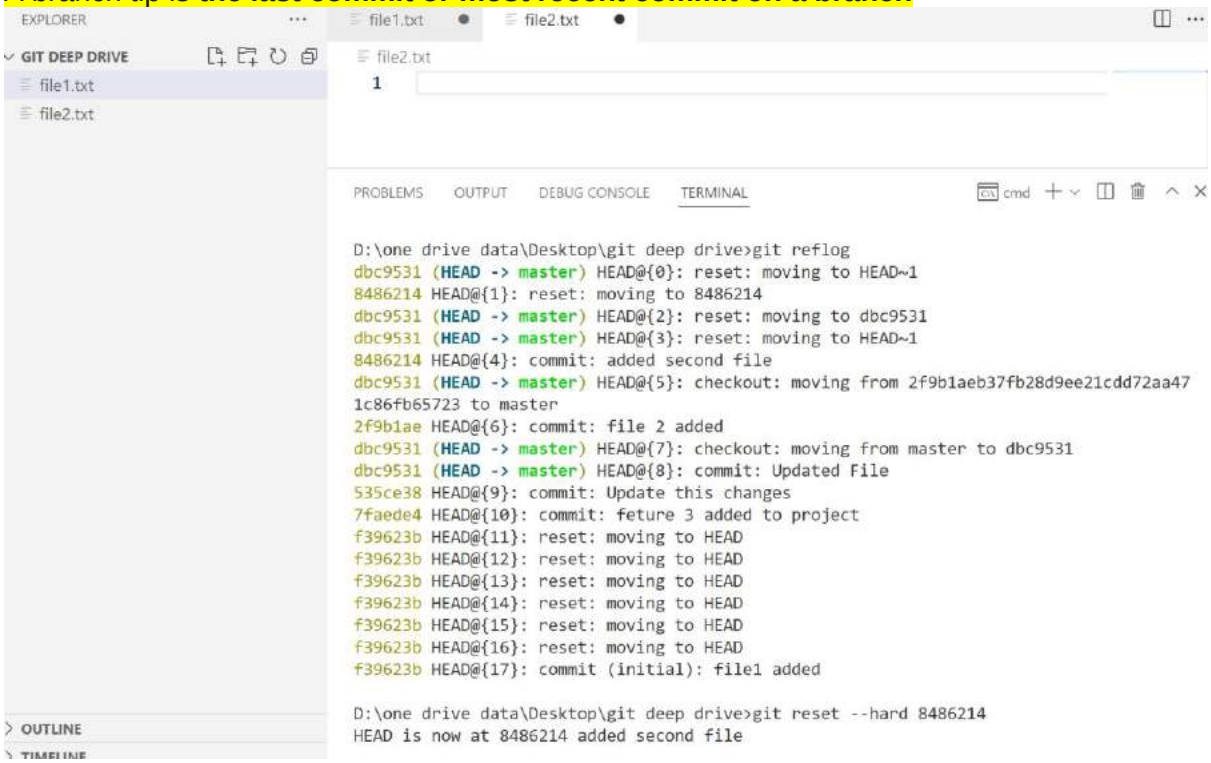


Fig. git reflog

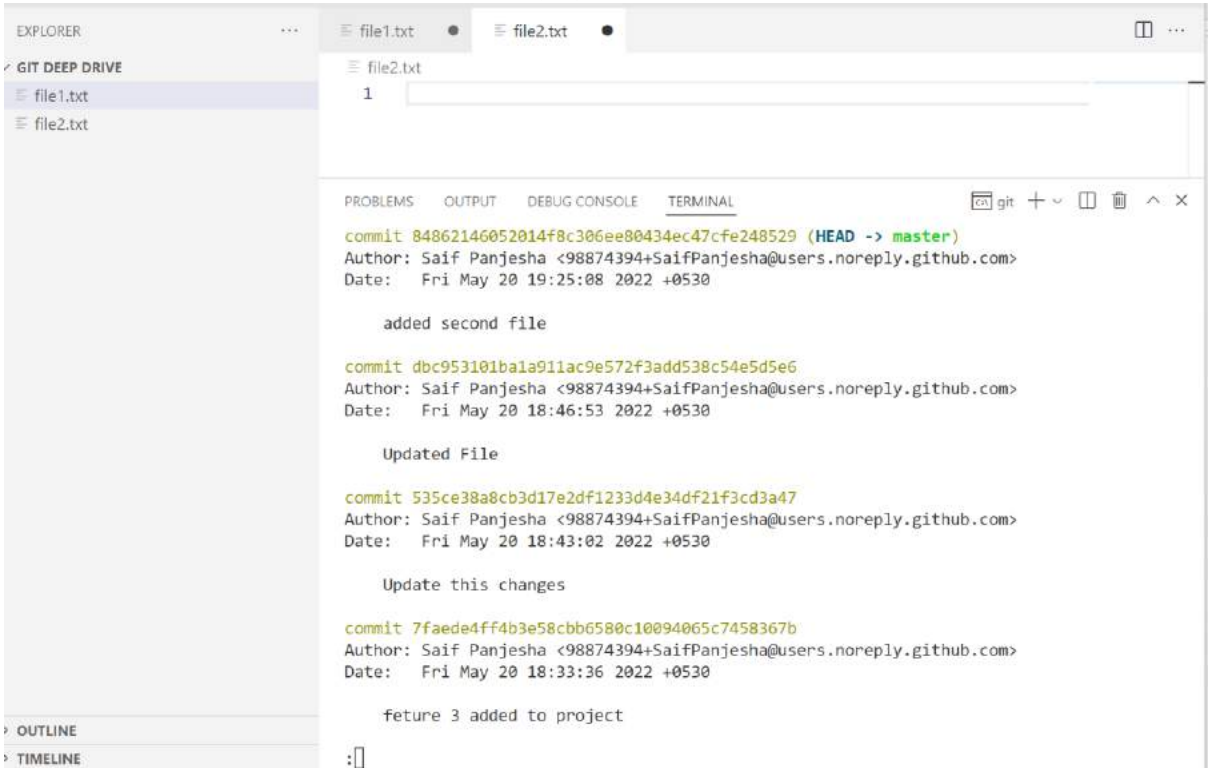


Fig. Successful added second file with commit history

“reflog” in Branches:

Let’s create a new branch called **feature_branch** and add file3.txt file

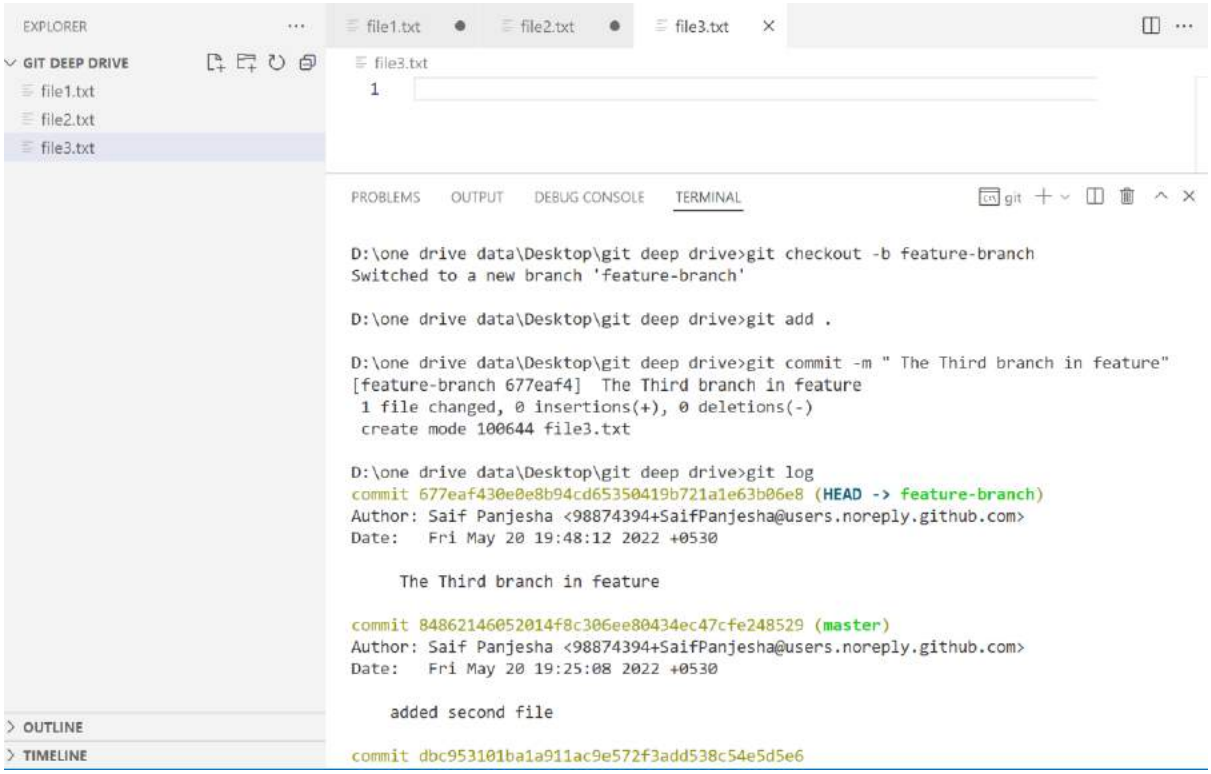


Fig. created a new branch with file3.txt

Switch back to master branch:

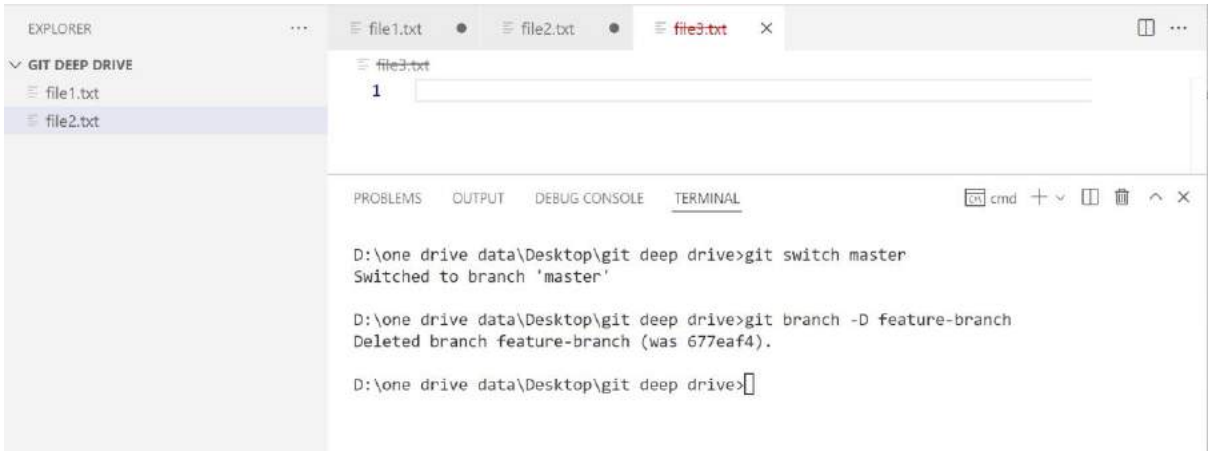


Fig. feature and file3.txt deleted

For restore the **“branches”** is differs from **“commit”**

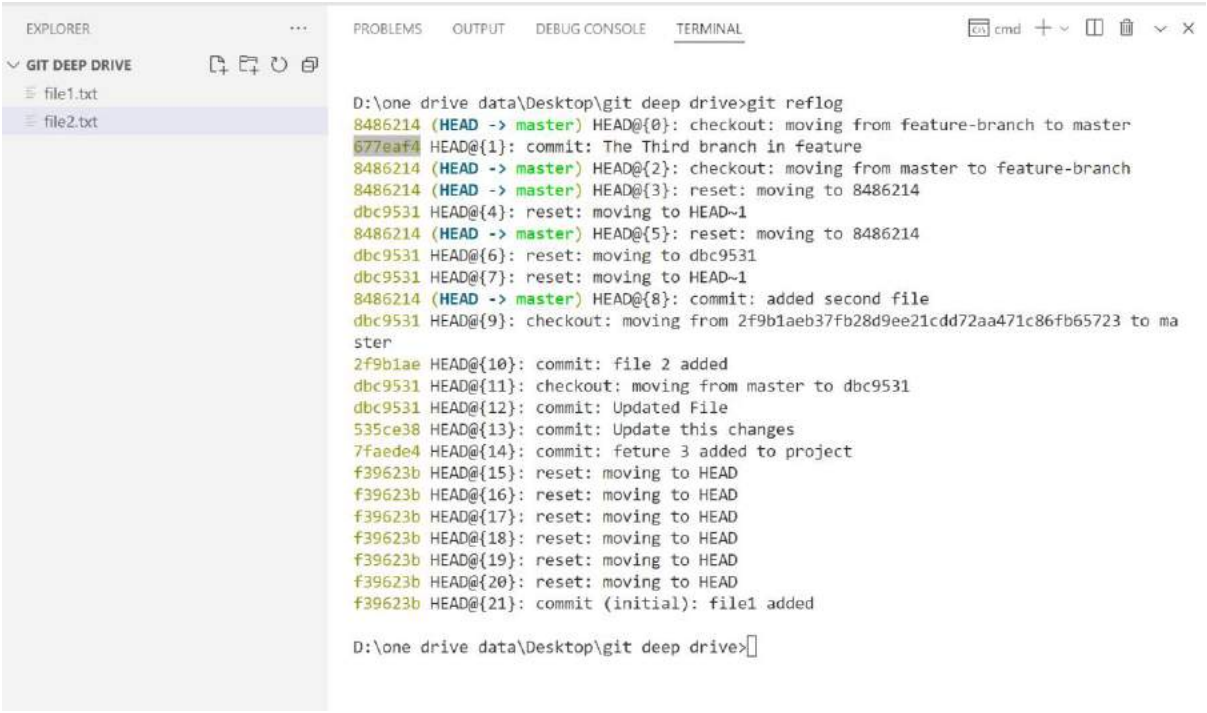


Fig. git reflog help us to get **Git keeps track of updates to the tip of branches**

Fig. updated track ID has added to detached head

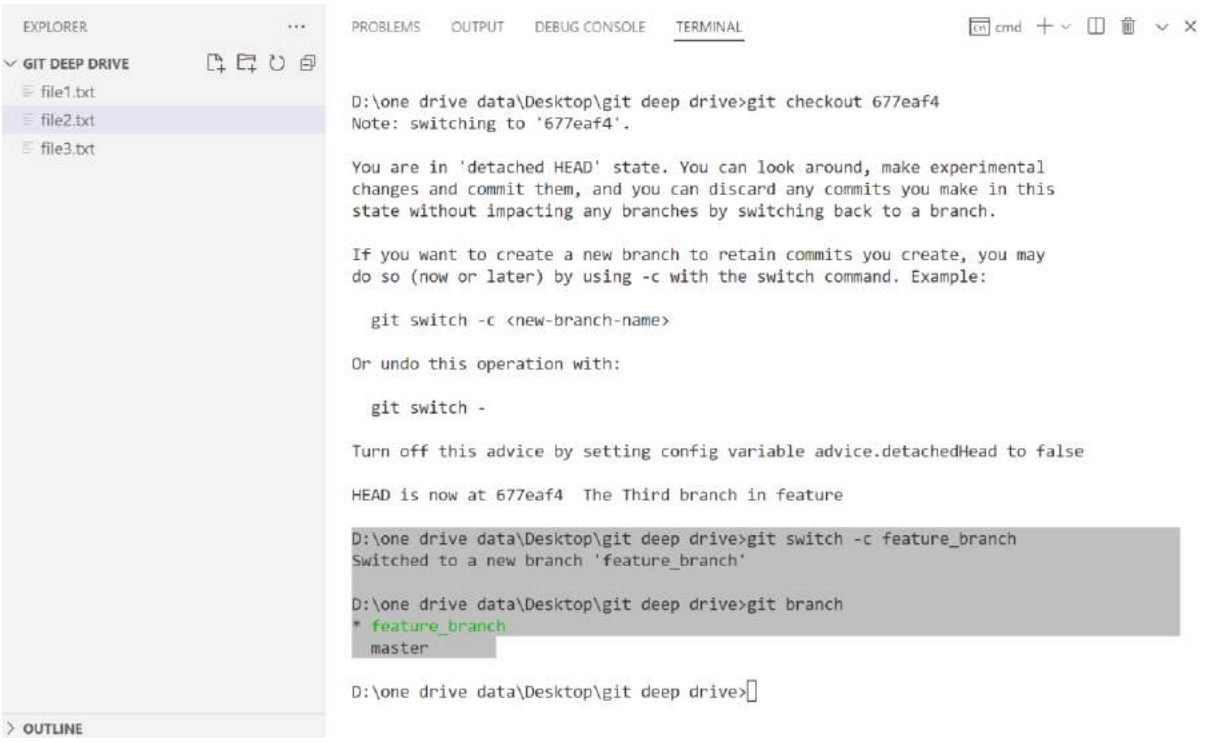


Fig. successful added branches with “reflog”

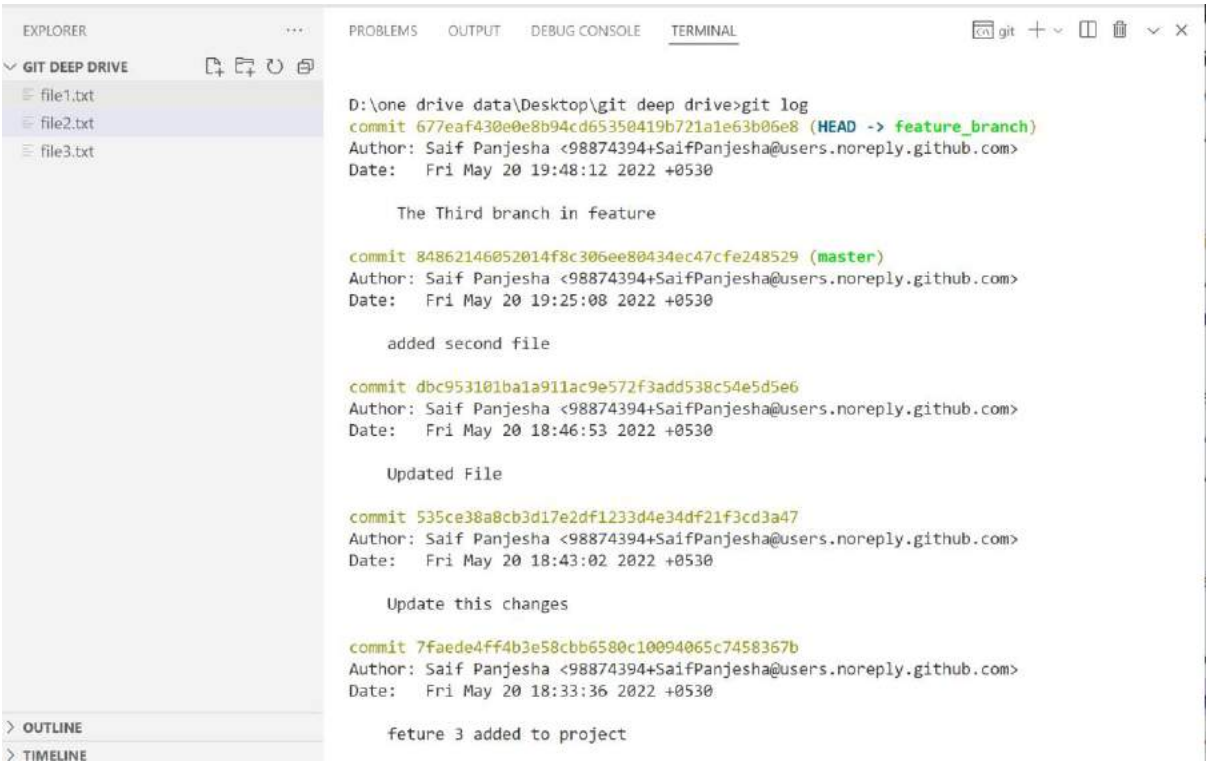


Fig. Commit history of current branch(feature_branch)

Combining Branches – What & Why?

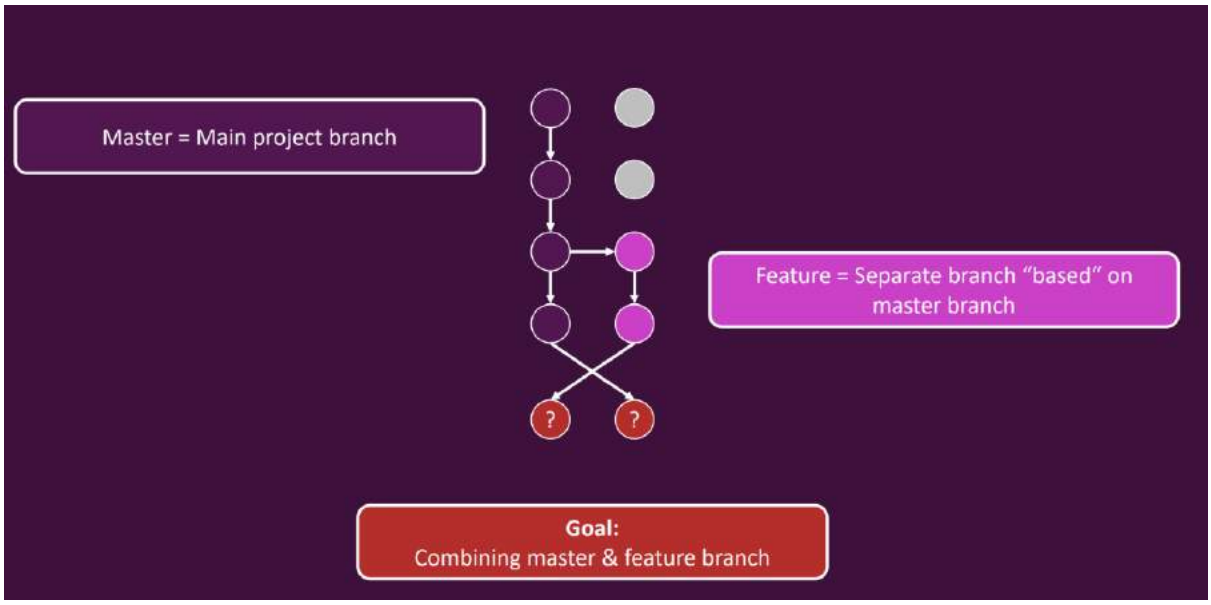


Fig. show Main Branch as Master and Separate Branch as feature

Understanding Merge Types:

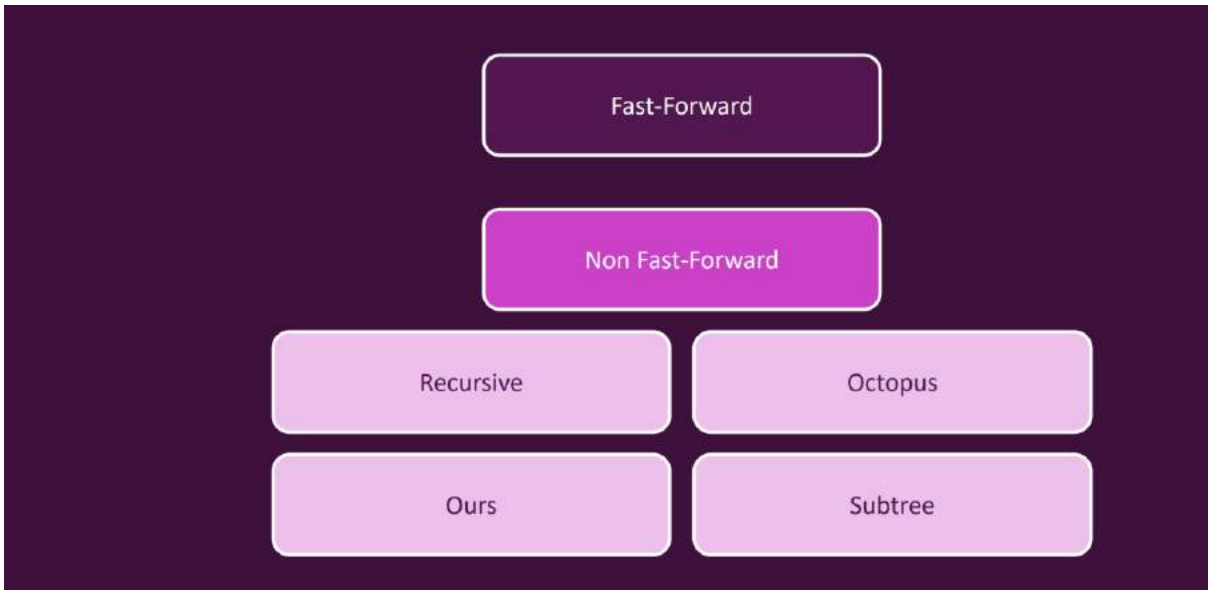


Fig. Shows Types of Merges in Git

Applying the Fast-Forward Merge:

Fast forward merge can be performed when there is a direct linear path from the source branch to the target branch. In fast-forward merge, git simply moves the source branch pointer to the target branch pointer without creating an extra merge commit.

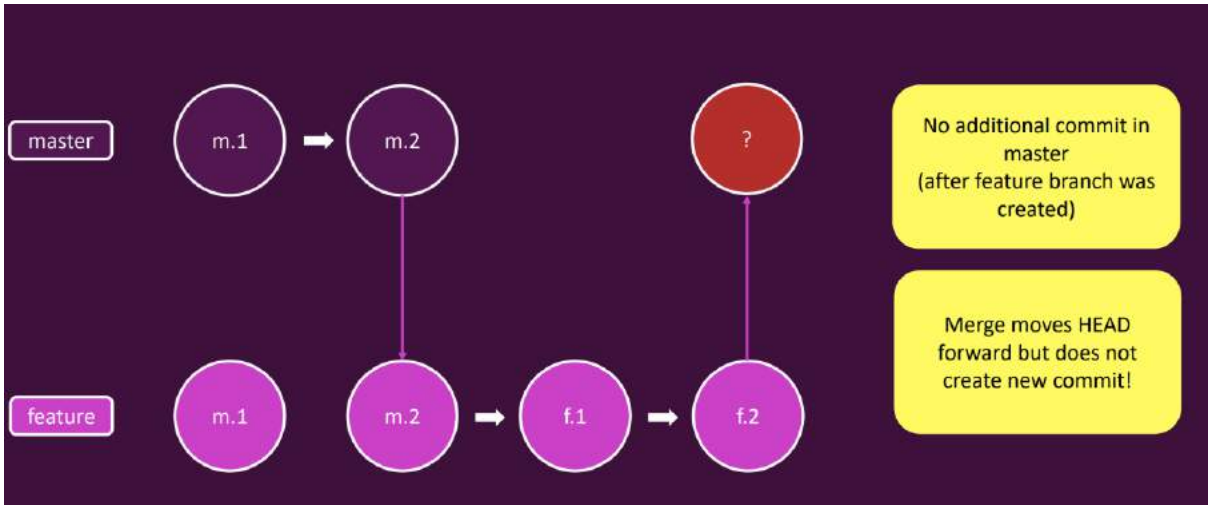


Fig. Shows Master & Feature -Merge("Fast-Forward")

Let's go and create first master branch with two files m1.txt first commit

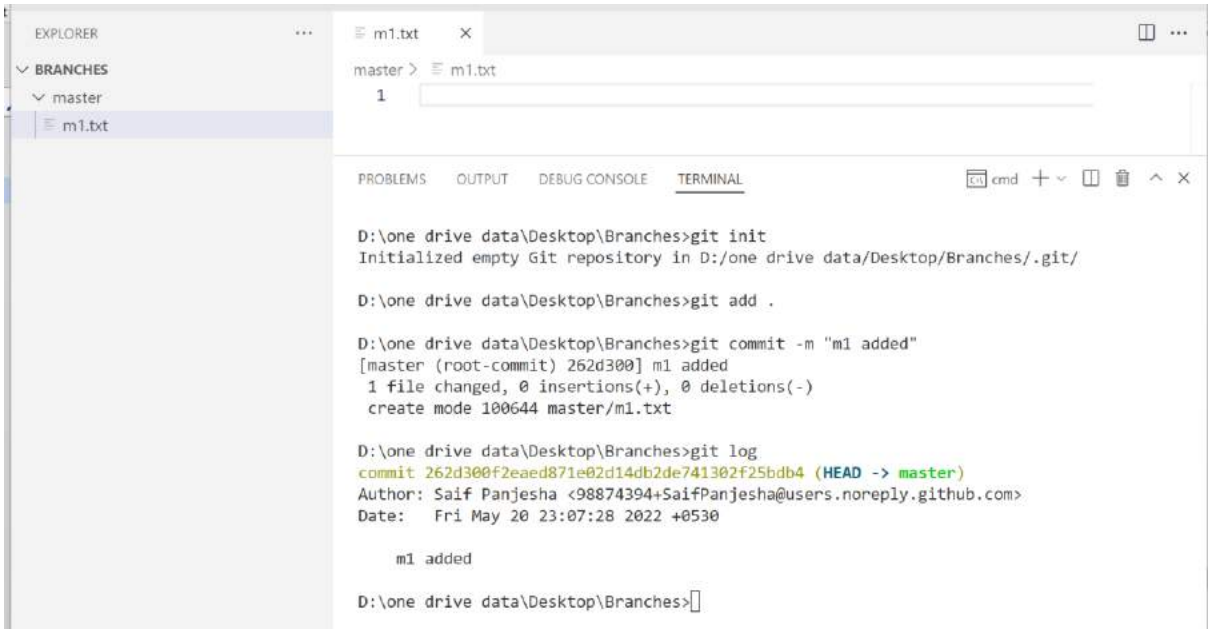


Fig. Created master branch with first commit

Let’s go and create first master branch with two files m2.txt second commit

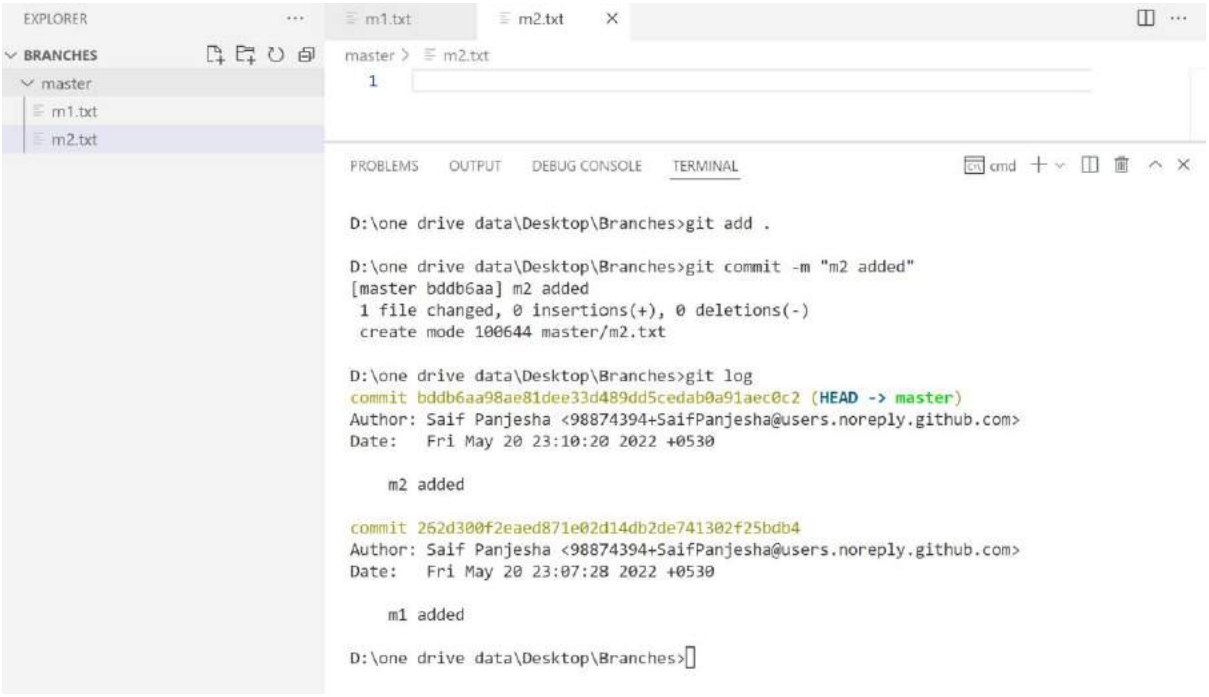


Fig. Created master branch with second commit

Now, lets create a new branch called **“Feature”** branch with two files f1.txt first commit and f2.txt second commit

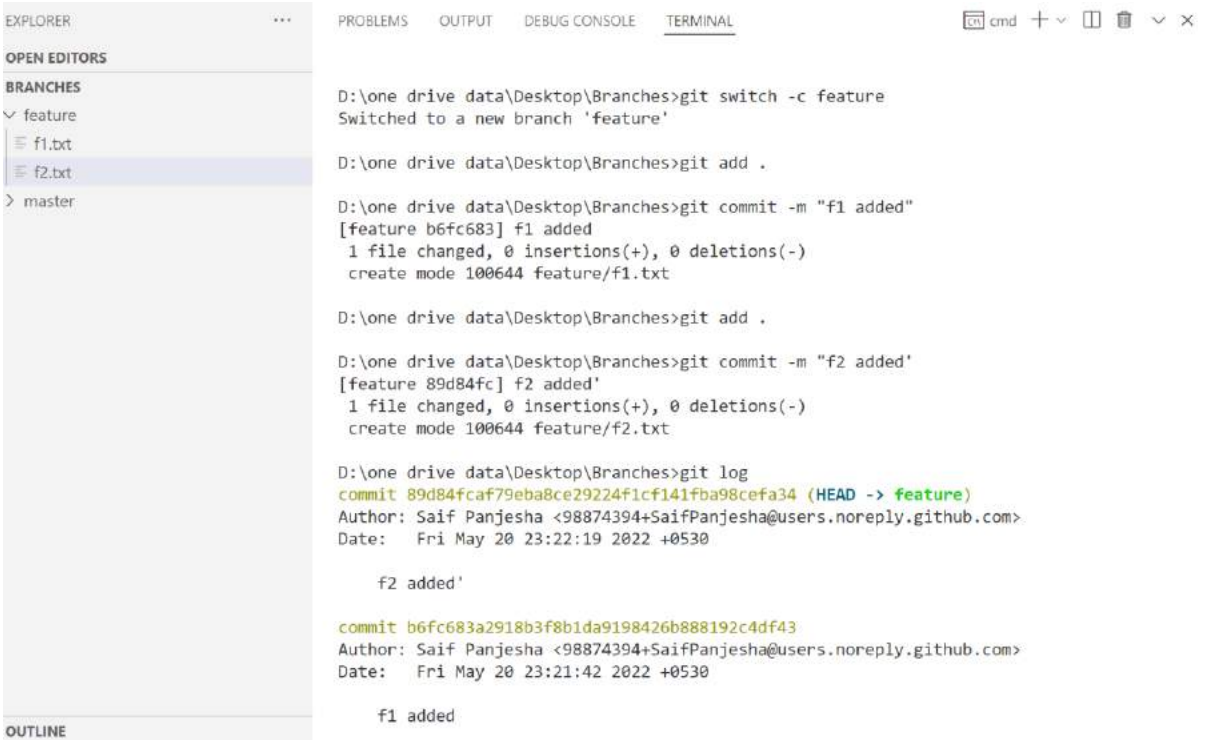


Fig. Created feature branch with two files f1.txt and f2.txt

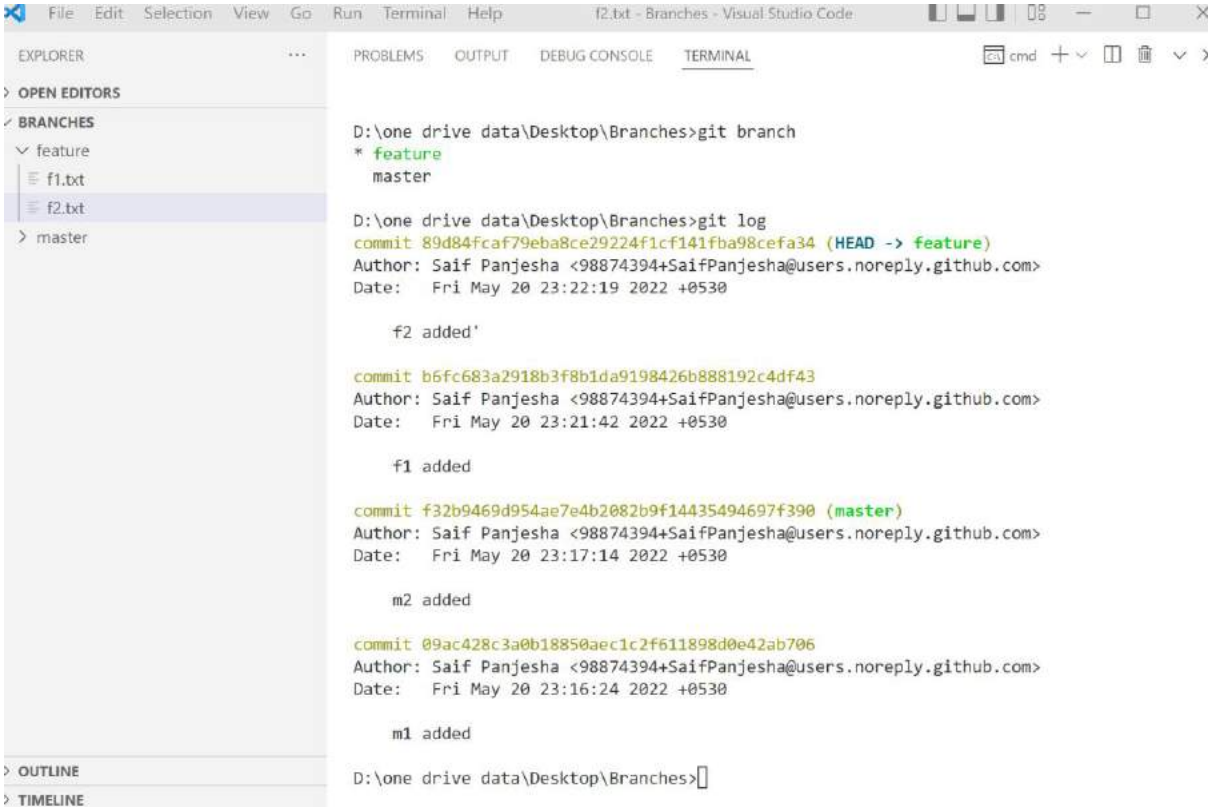


Fig. Created feature branch with two files and checking commit history

Fig. Switched to master branch

Fast forward merge happens:

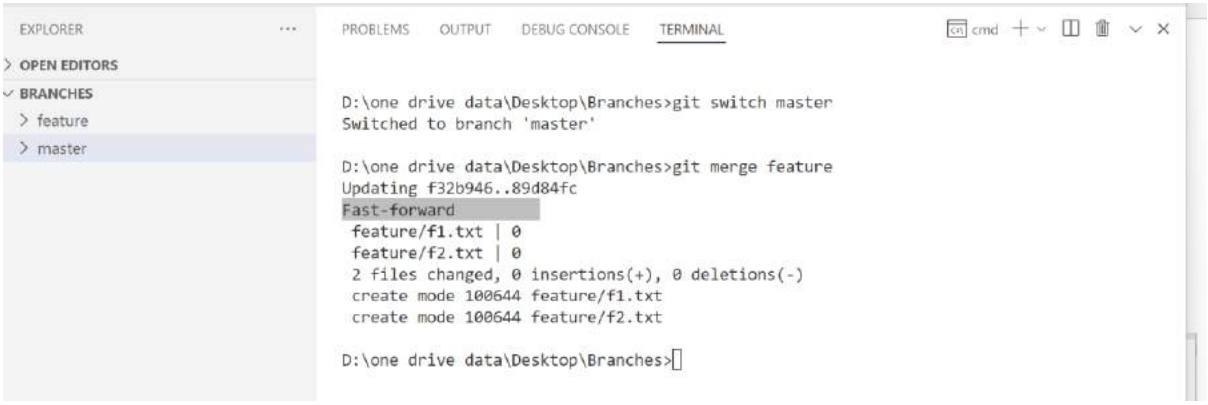


Fig. shows merged with master branch as fast - forward

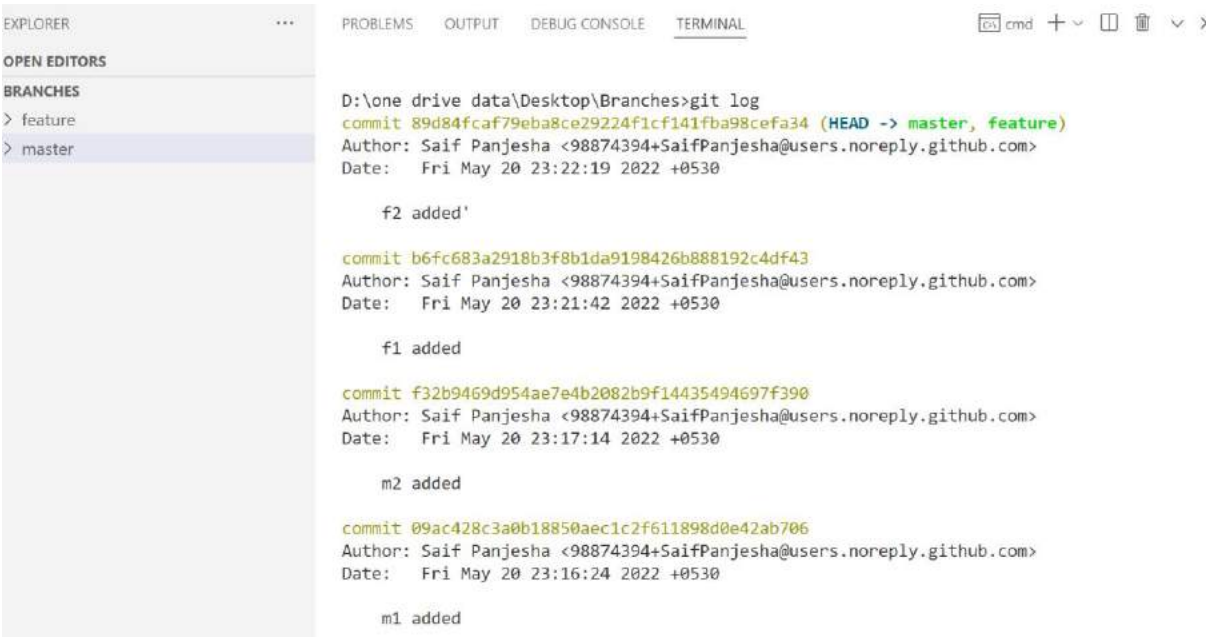
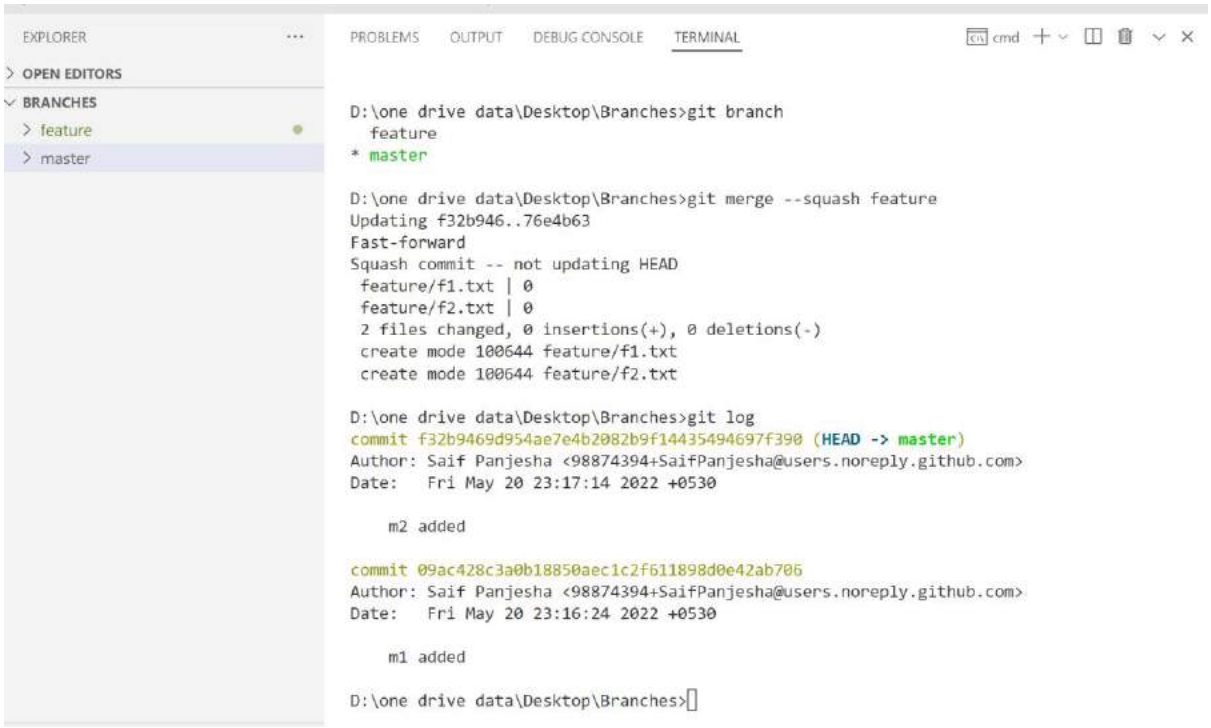


Fig. direct linear path from the source branch to the target branch

Fig. reset the head to remove extra commit

Note:

Squash will simply well, kind of squash or put together all the commits we had in our feature branch into the latest commit so to say. So only one commit is added to our master branch in the end.Therefore, if we now use git merge --squash feature, you can see that we still have a Fast-forward merge here but if we now check our Git log,
you see well, we don't have any commit, right?Now, what's wrong here?
Well, nothing is wrong. If you quickly scroll up a bit, you see that the head was not updated here
because as I said, with the squash command, with the squash flag, we will put together all the changes made in the feature branch into one single commit
and this commit is not our f2 commit here, it is, in the end, as it contains all the changes,but we have to create a separate commit



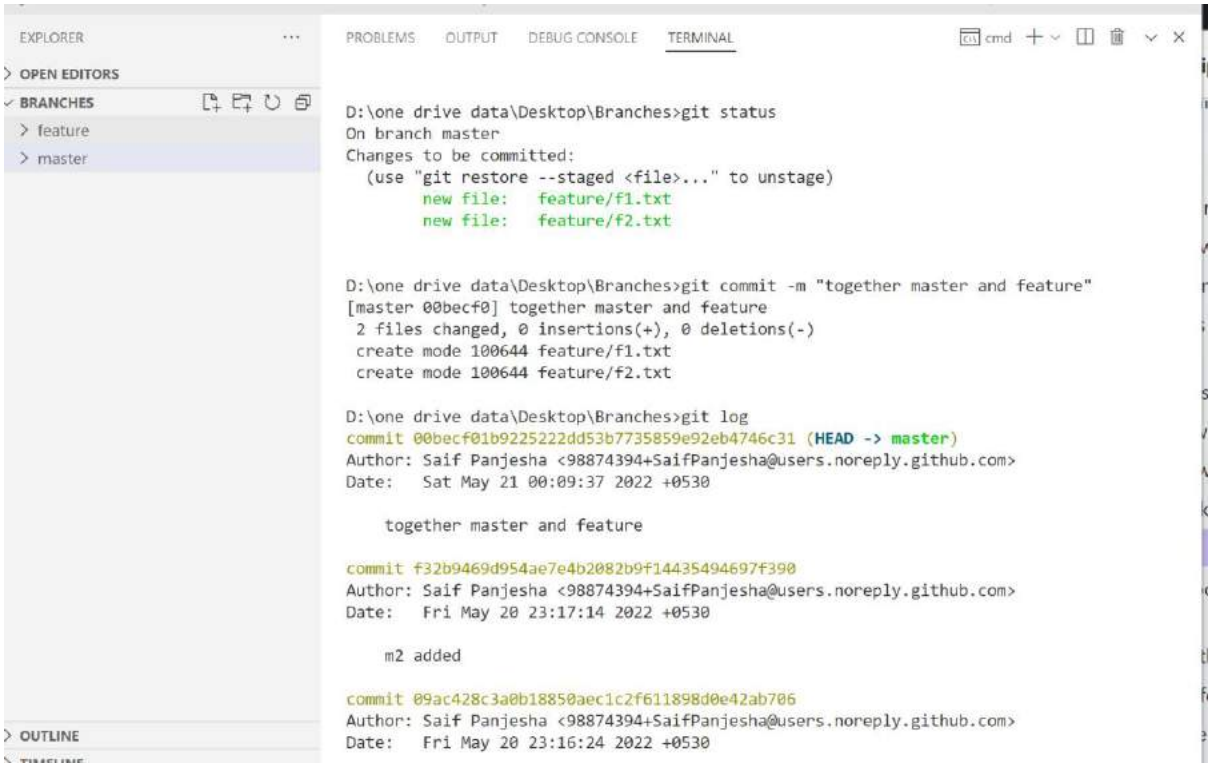


Fig. squash the previous commits into one.

Cleaning the directory:

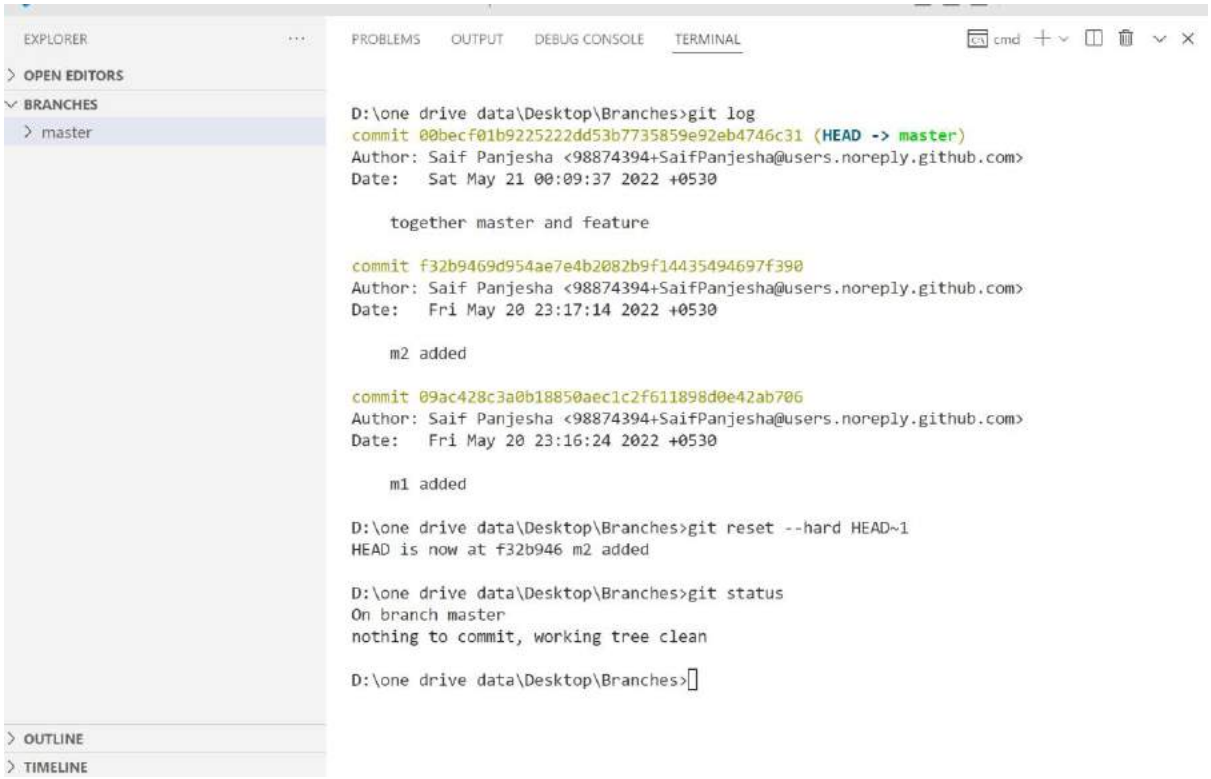


Fig. Cleaning the directory.

The Recursive Merge (Non – Fast Forward):

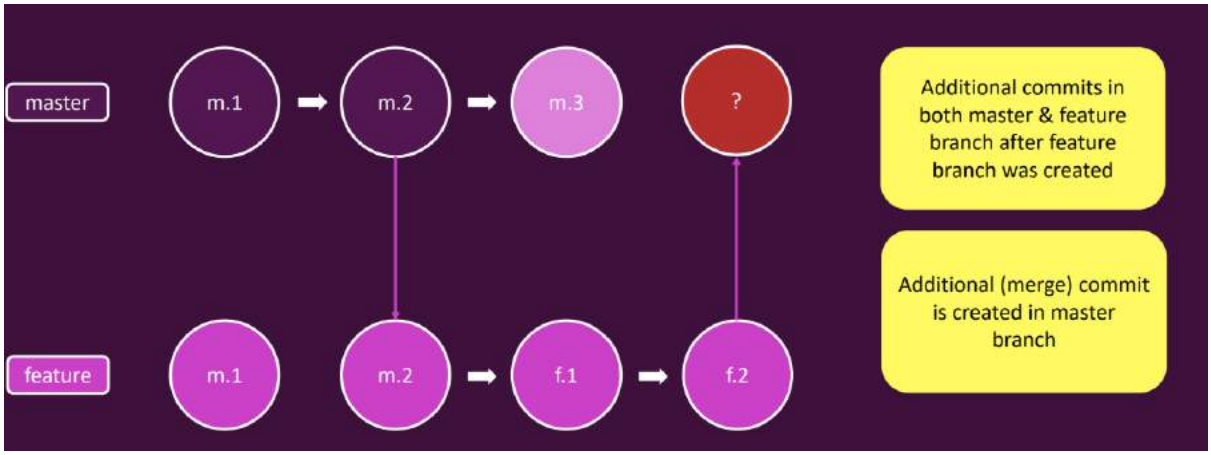


Fig. Shows Master & Feature – Recursive Merge(“Non -Fast-Forward”)

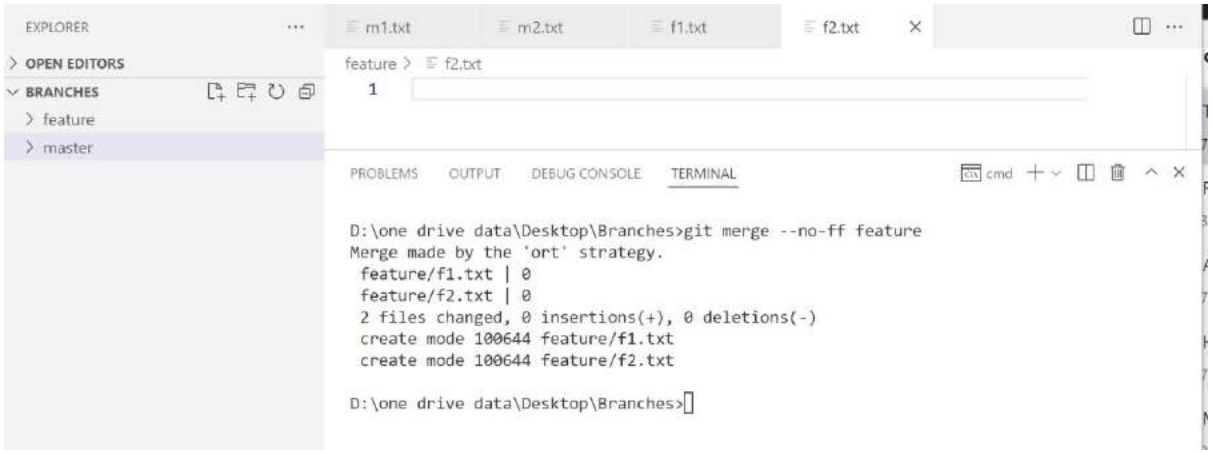


Fig. recursive merge has been added git merge- - no-ff feature

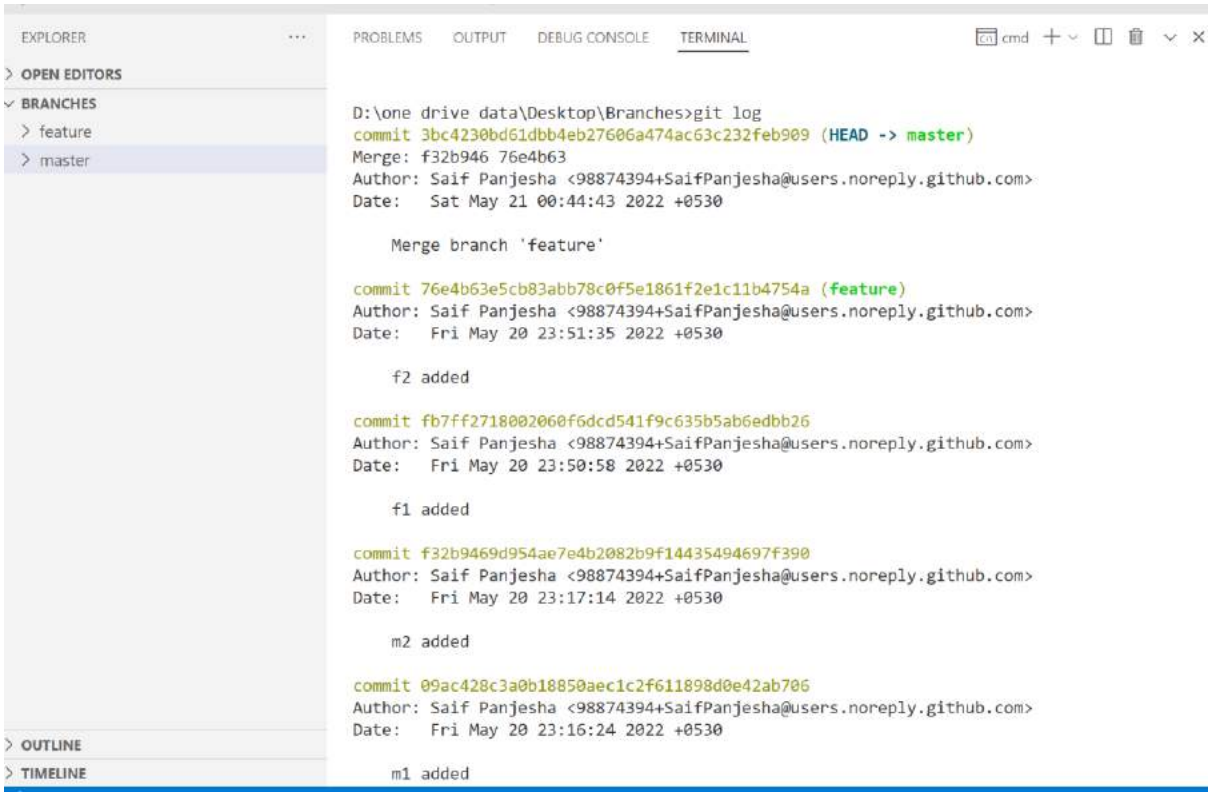


Fig. Shows Complete history on the master branch of feature latest commit (Merge branch “feature”)

We can’t go back to previous commit

D:\one drive data\Desktop\Branches>git reset --hard HEAD~3

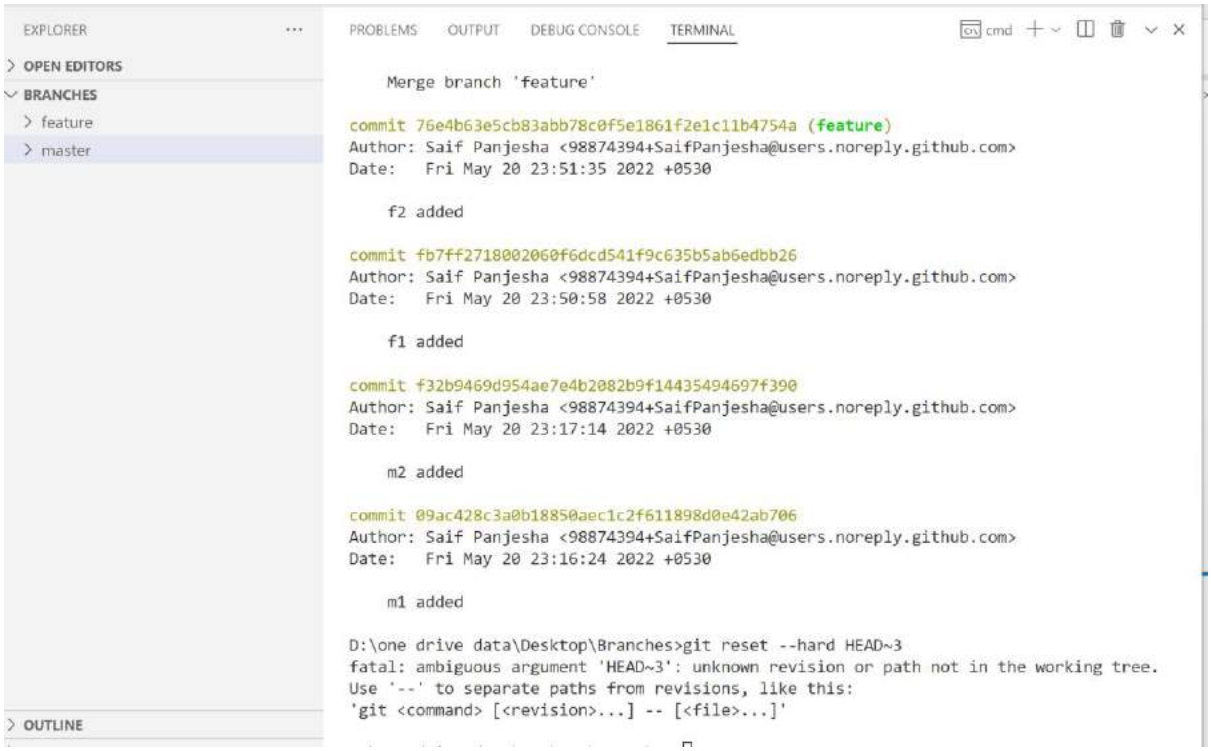


Fig. Fatal Error

To resolve this error, we need to undo the latest commit only:

D:\one drive data\Desktop\Branches>git reset --hard HEAD~1

Fig. Error resolved

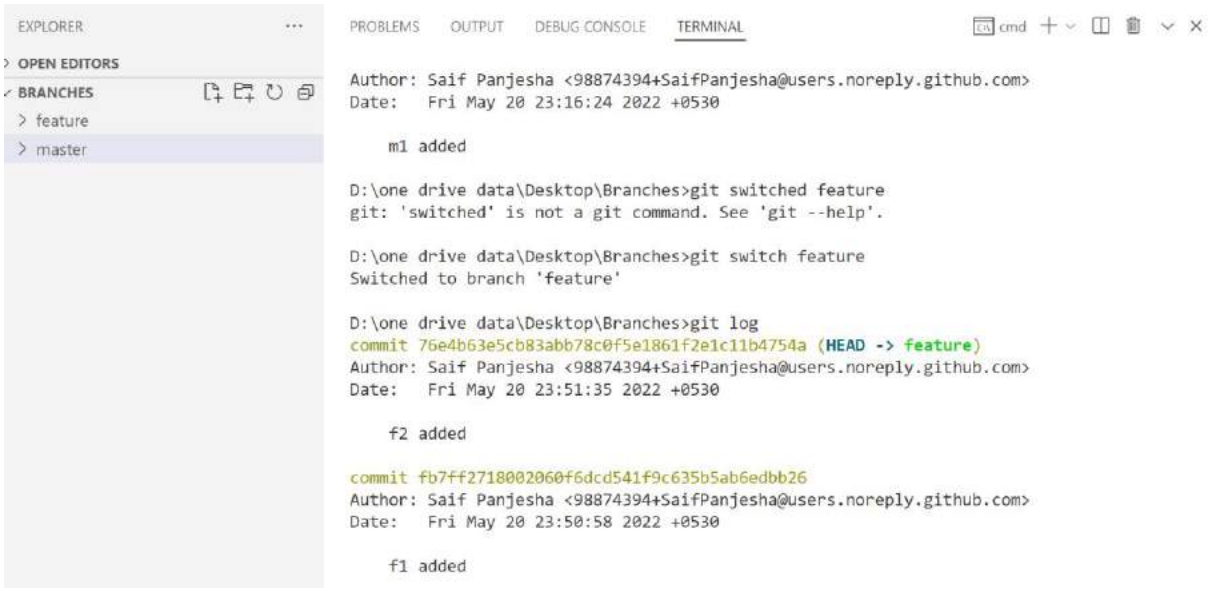


Fig. feature switched commit history is same in current branch

Let's Switch to master and create m3.txt file

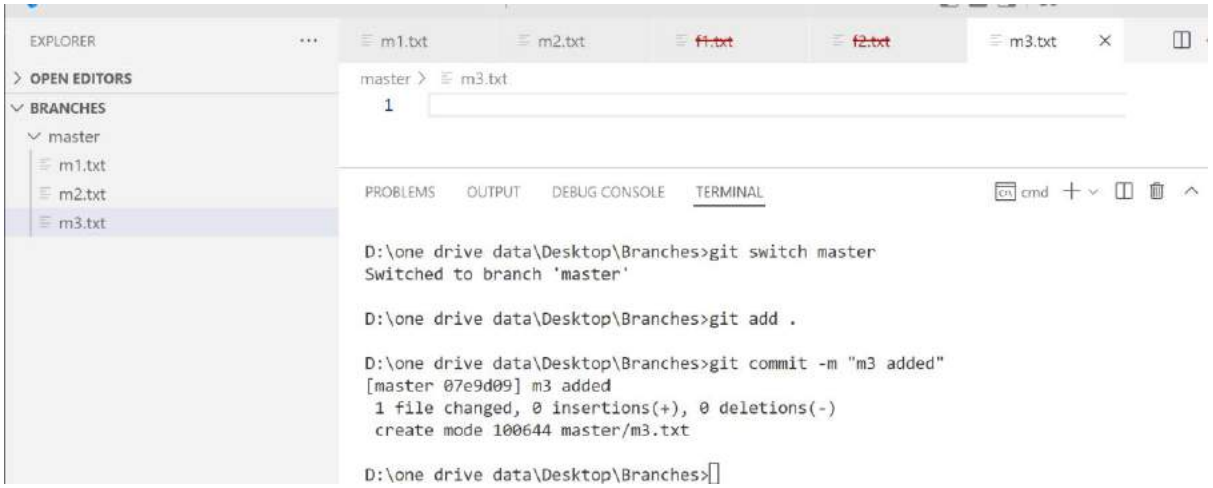


Fig. Successful created m3.txt file

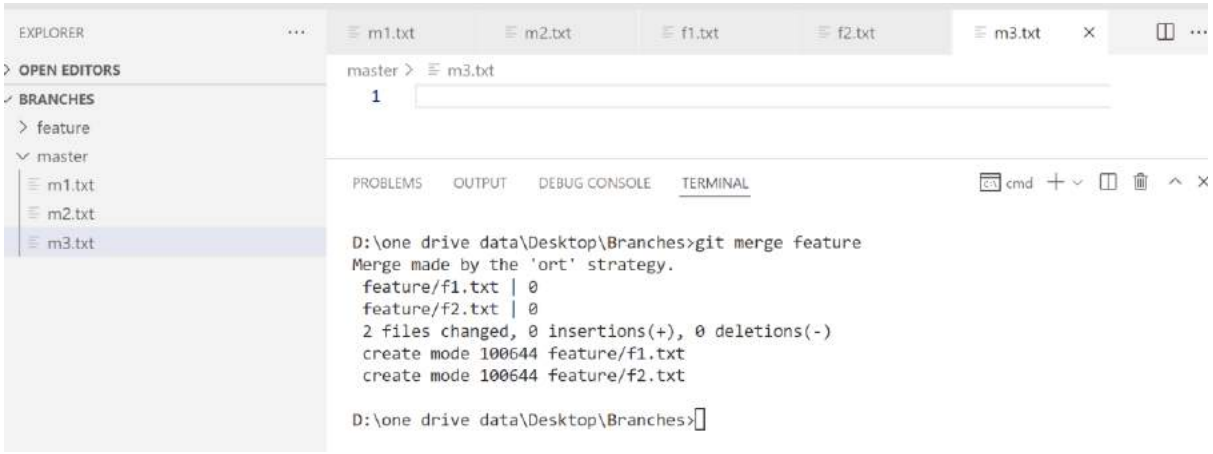


Fig. Successful merge feature branch into master

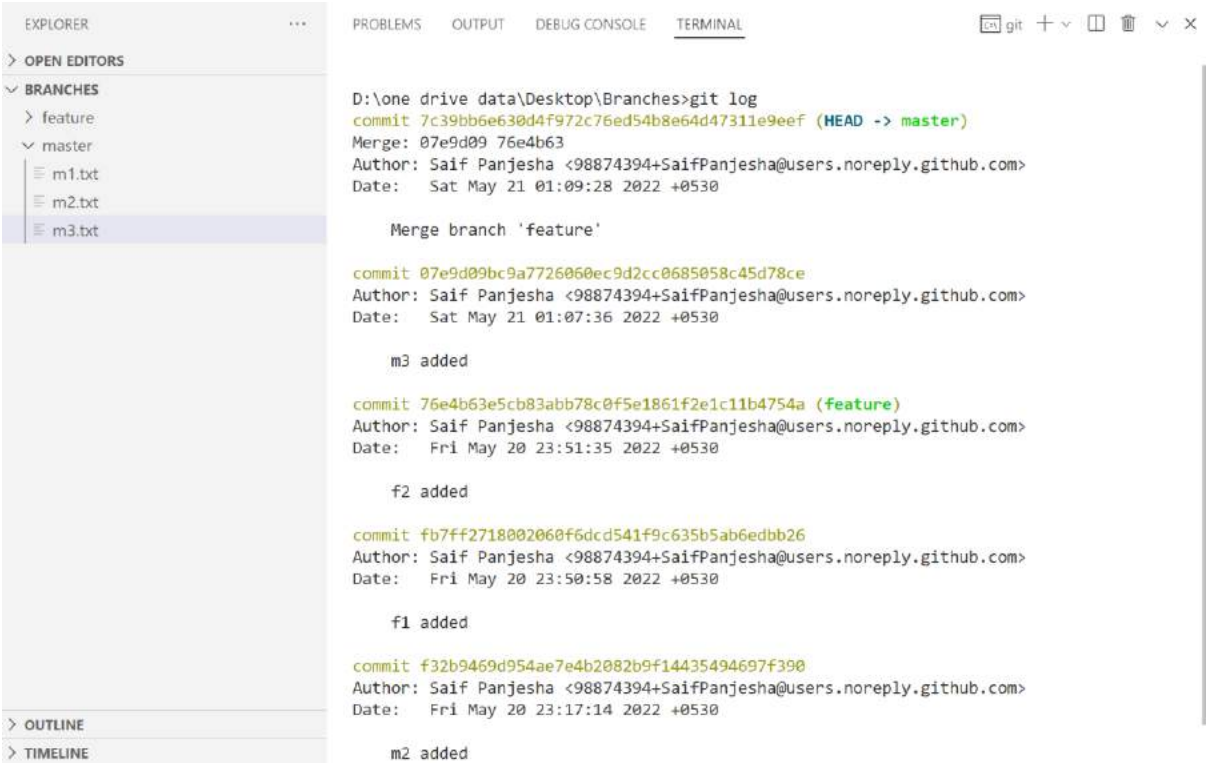


Fig. full commit history shows in master branch

Let's do again –squash

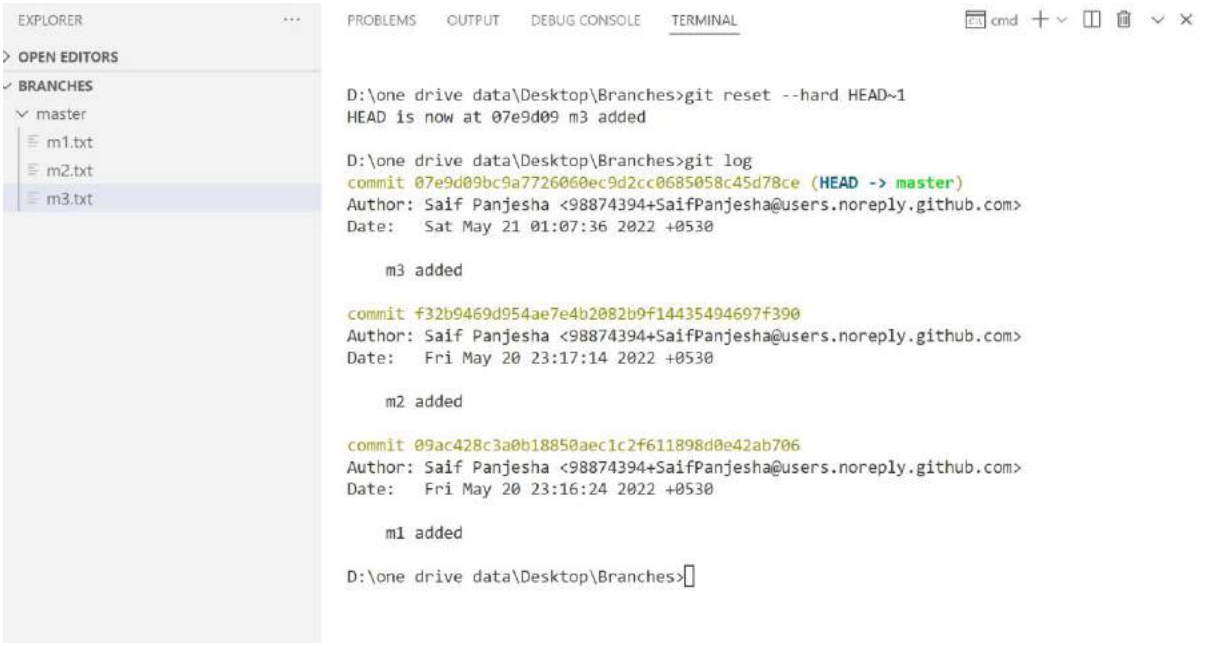


Fig. switch to previous commit

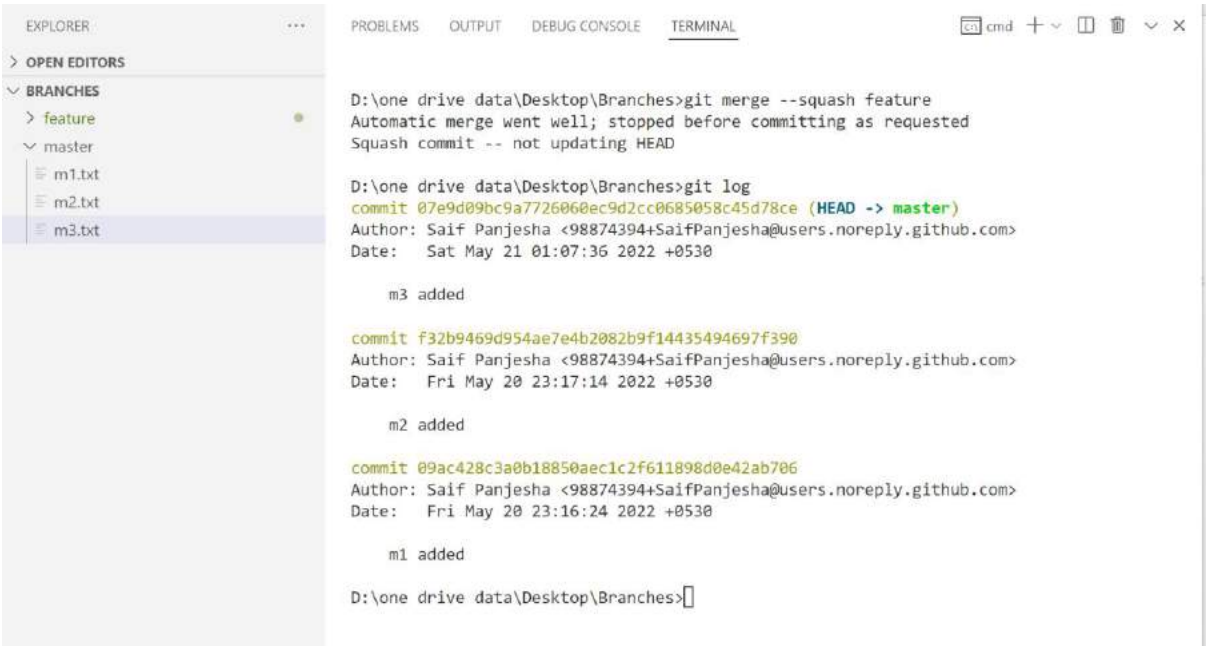


Fig. squash is used to squash the previous commits into one

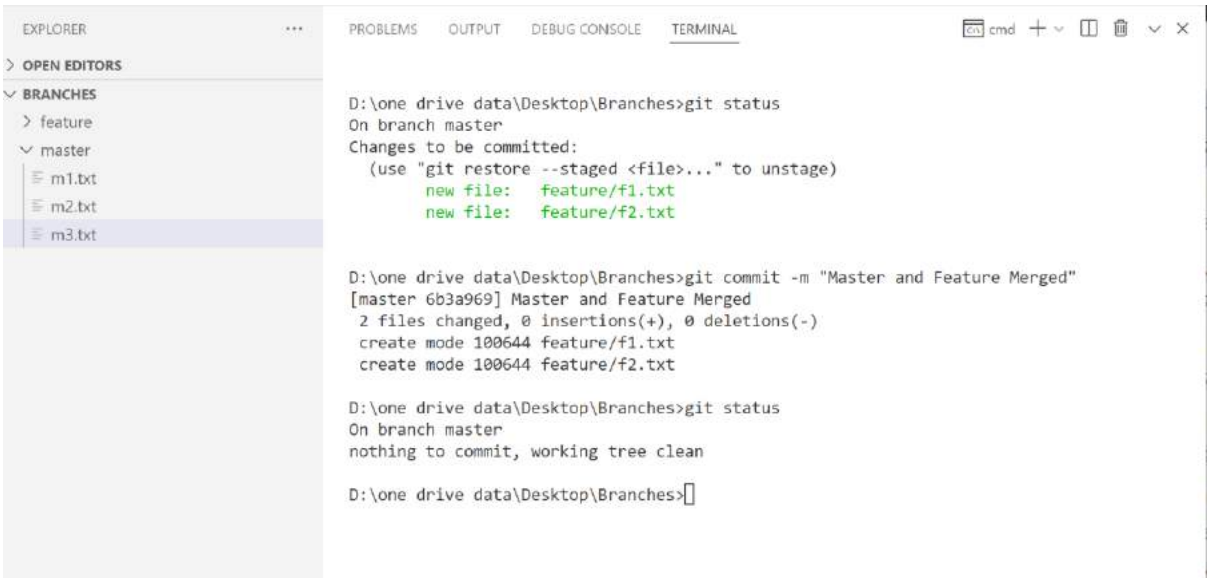


Fig. Committed Master and Feature Merge

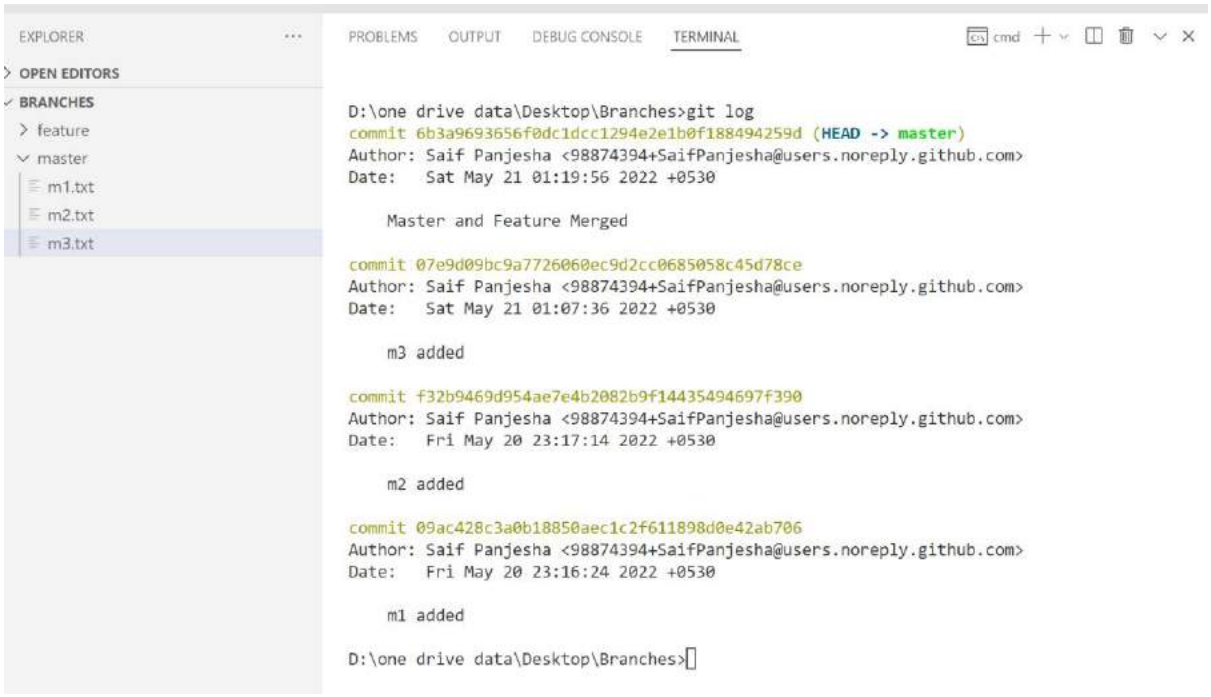


Fig. Success Merged in Commit History

Rebasing Theory:

Rebasing is a process to reapply commits on top of another base trip.

It is used to apply a sequence of commits from distinct branches into a final commit.

It is an alternative of git merge command

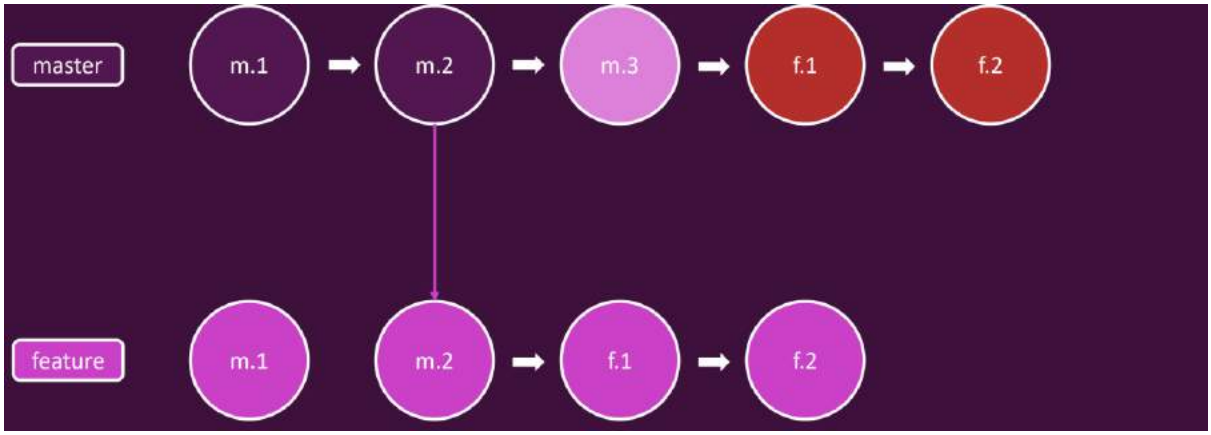


Fig. Shows Master & Feature of rebase

What Happens? When we use rebase??

From a content perspective, rebasing is changing the base of your branch from one commit to another making it appear as if you'd created your branch from a different commit.

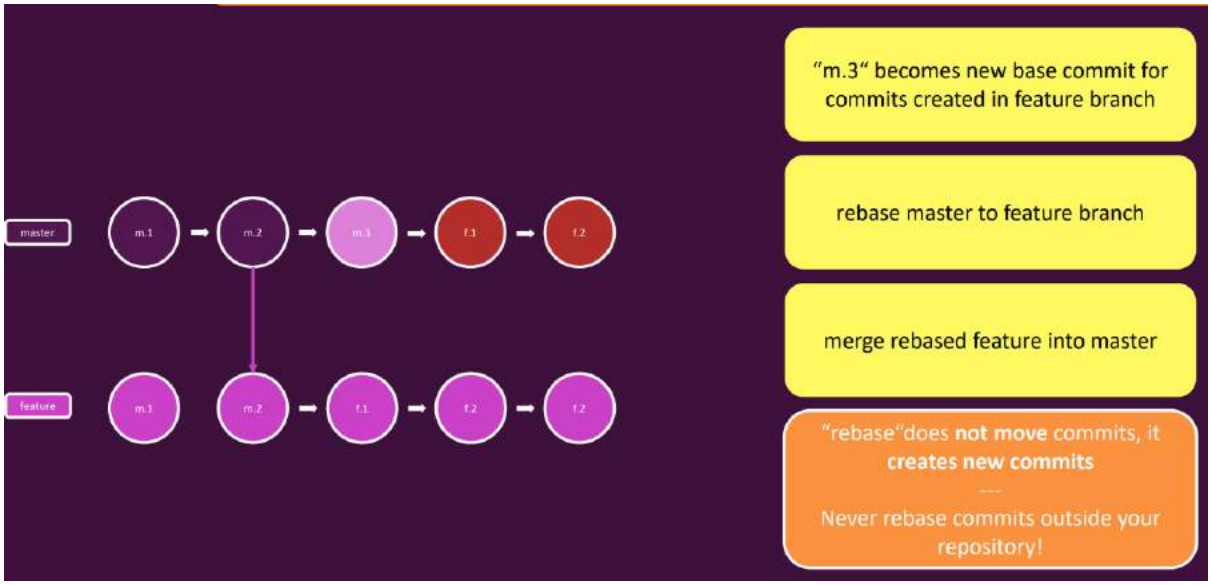


Fig. Shows using of rebase

Let's start rebasing

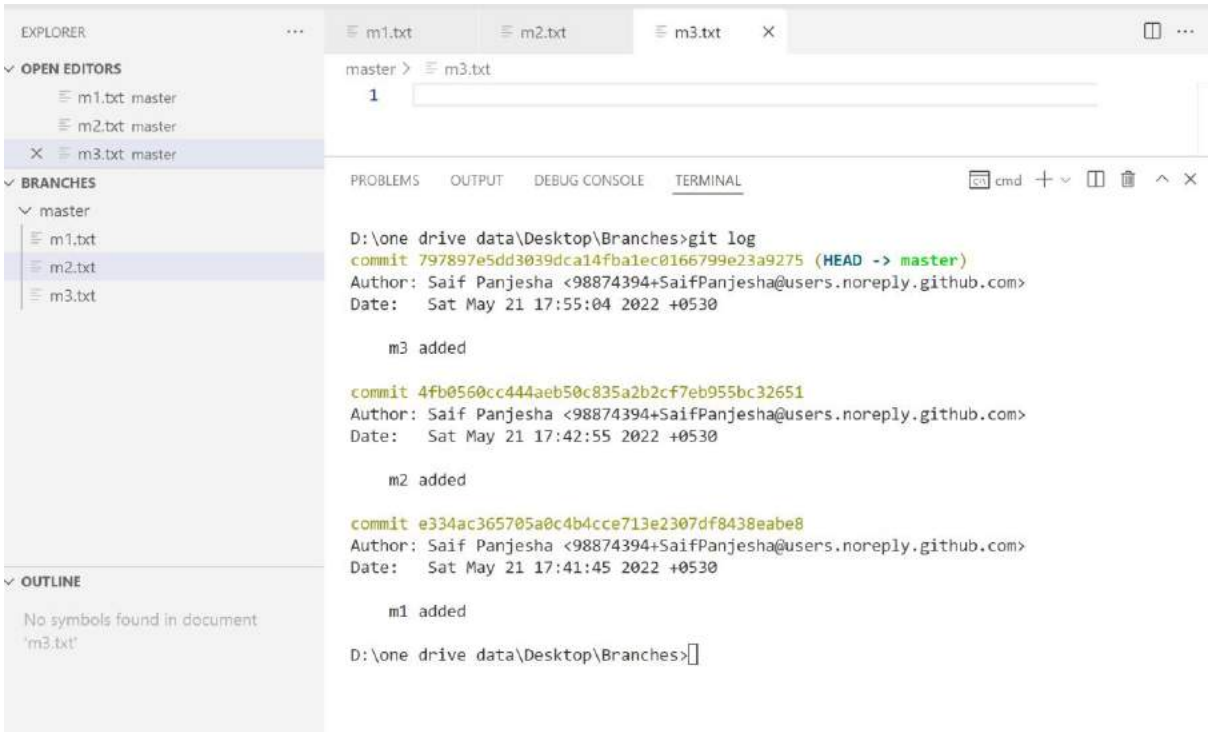


Fig. Exactly same structure as shows rebase of master as above referenced

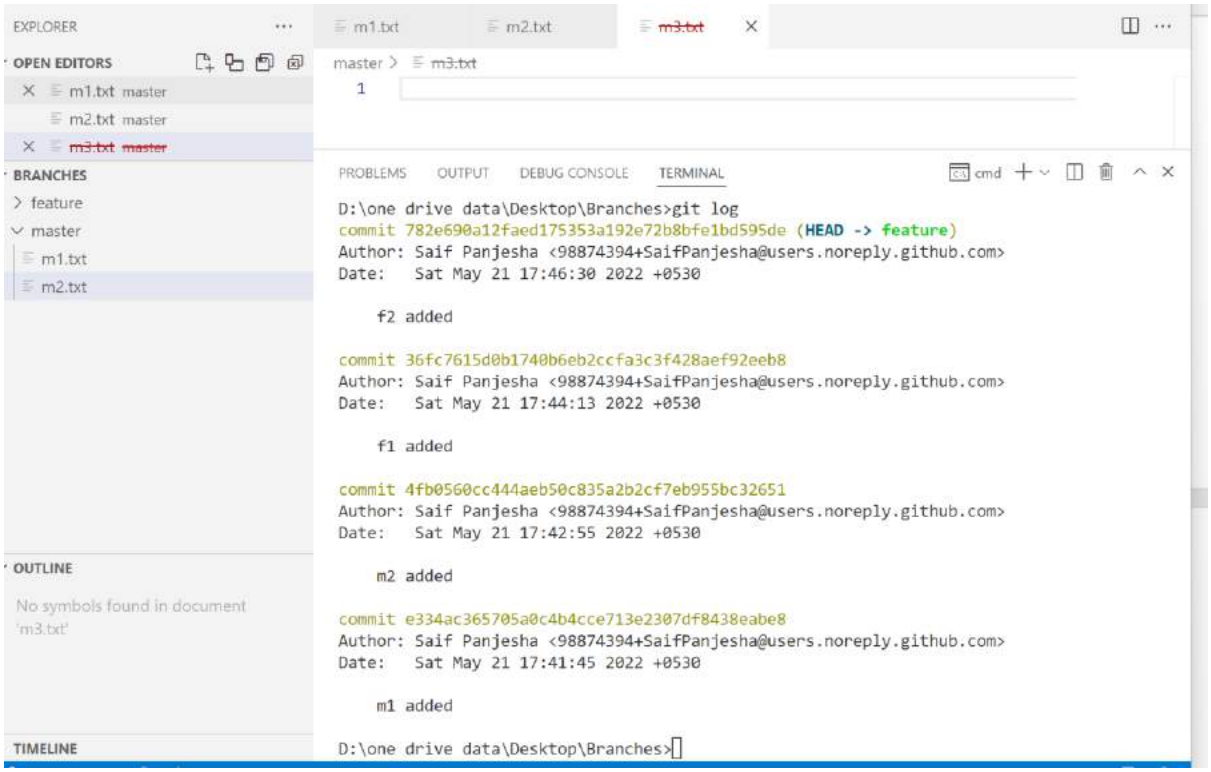


Fig. Exactly same structure as shows rebases of feature as above referenced

Applying rebase in feature branch

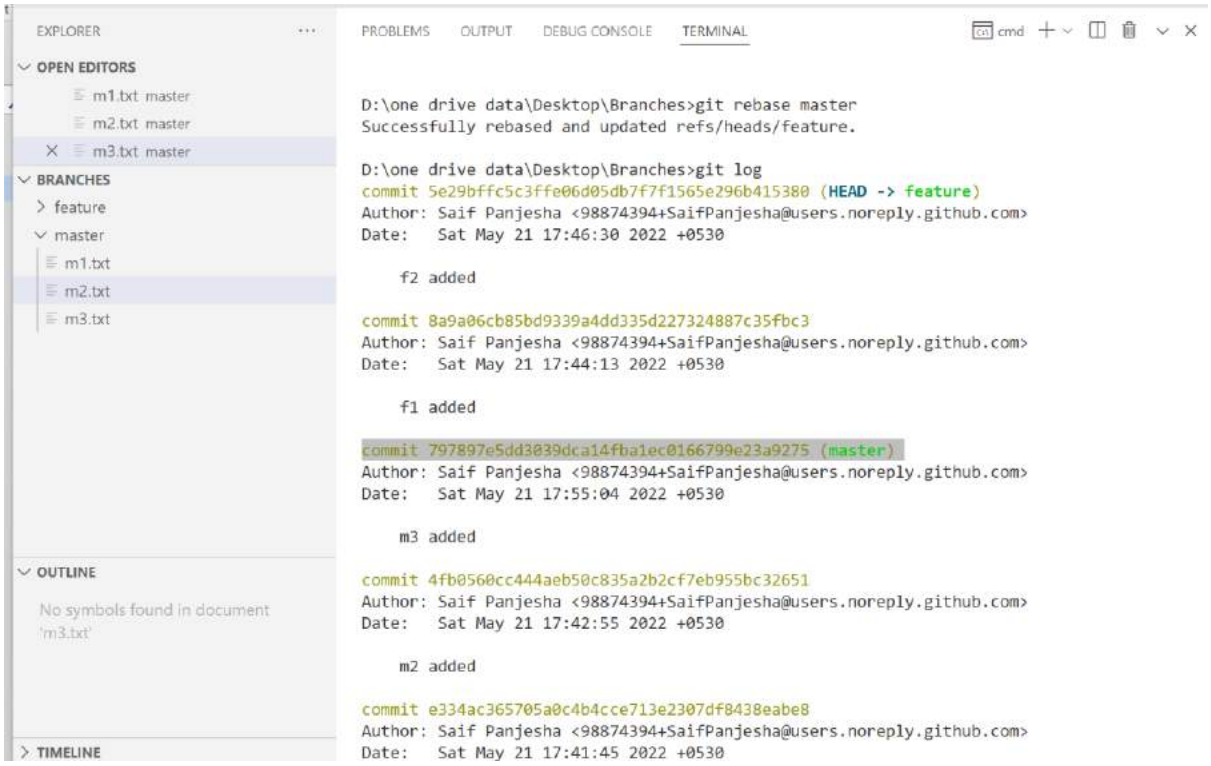


Fig. Successful applied rebase in feature branch to merge master commits.

Remember Note: after applying rebase commit history ID changed

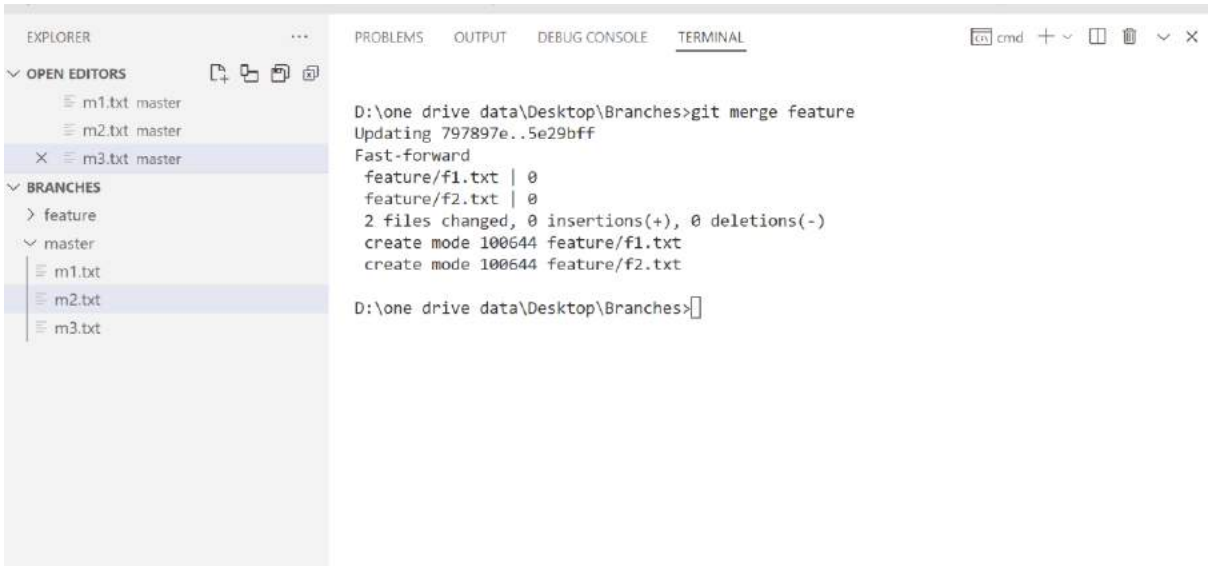


Fig. fast-forward merging

When to apply rebase?

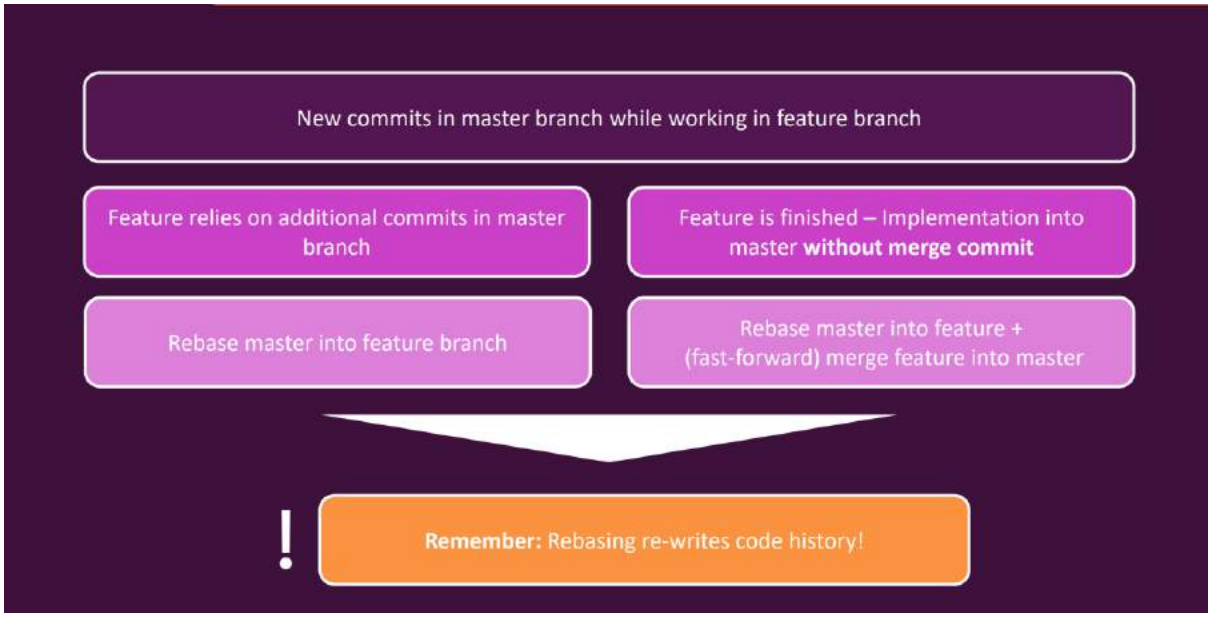


Fig. Workflow

Clever Note: **Never use rebase when project is dependent to others branches.**

Handling Merge Conflicts:

How to fix conflicts during the merge?

Scenario: If two people in two different branches work on same file in the end

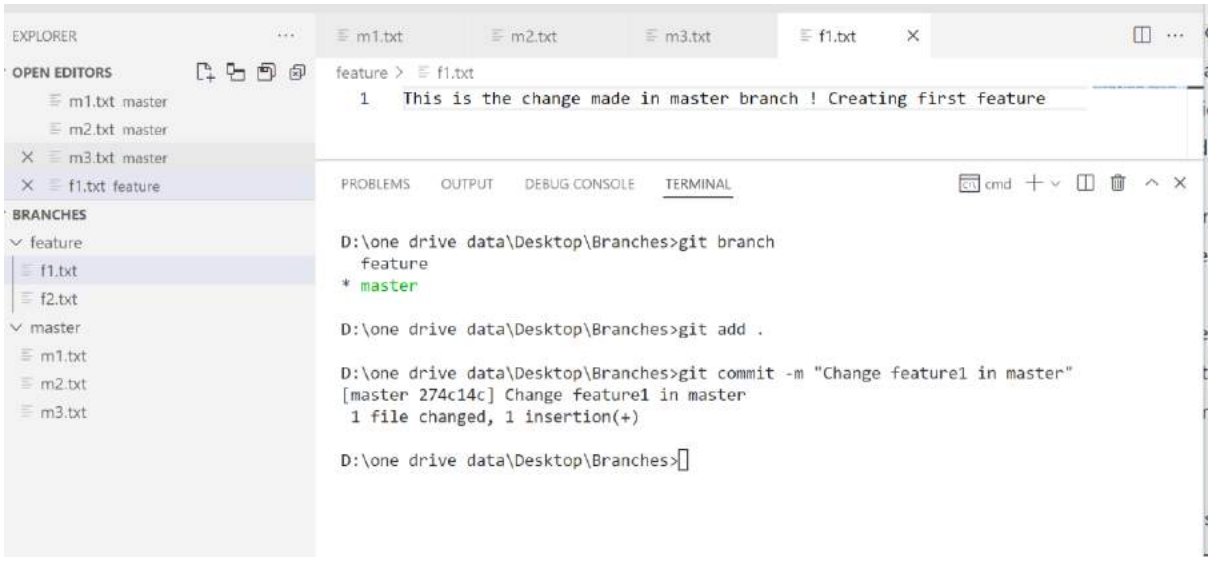


Fig. Shows Working feature in Master Branch

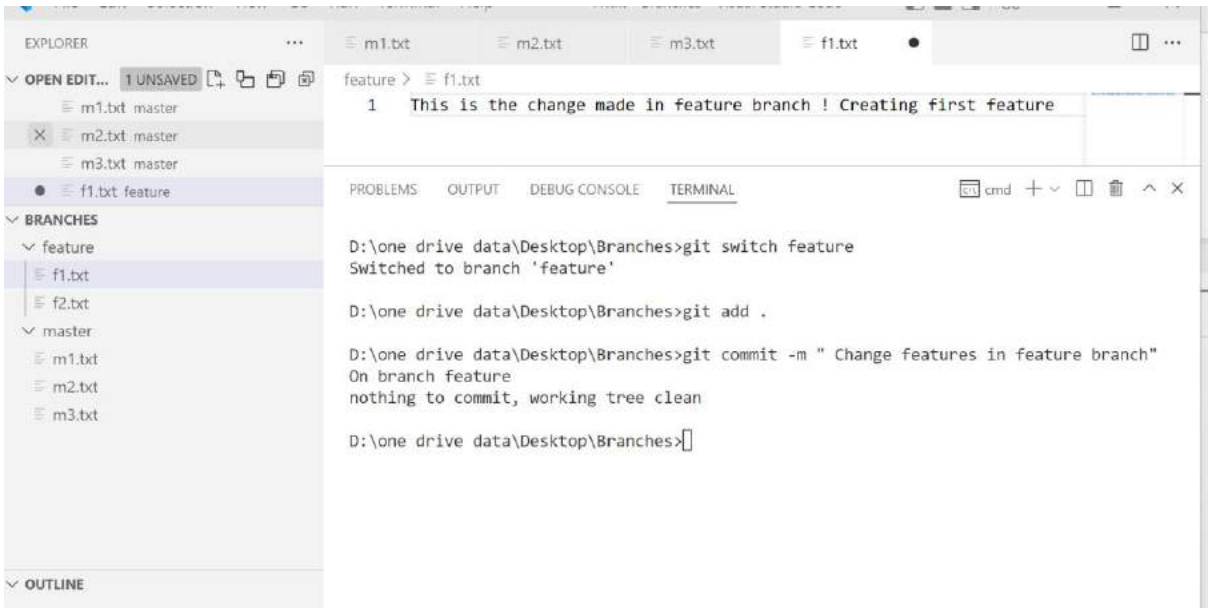


Fig. Shows Working feature in Feature Branch

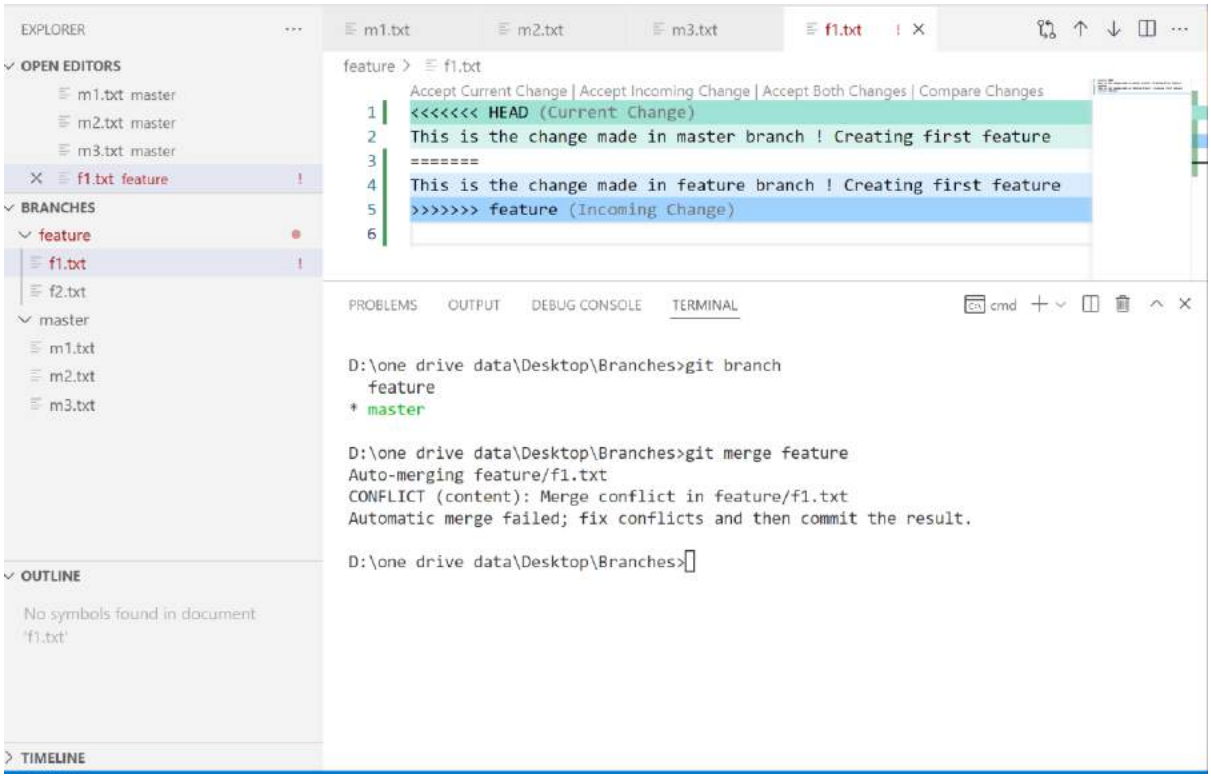


Fig. Shows Conflicts Created

How to handle these conflicts?

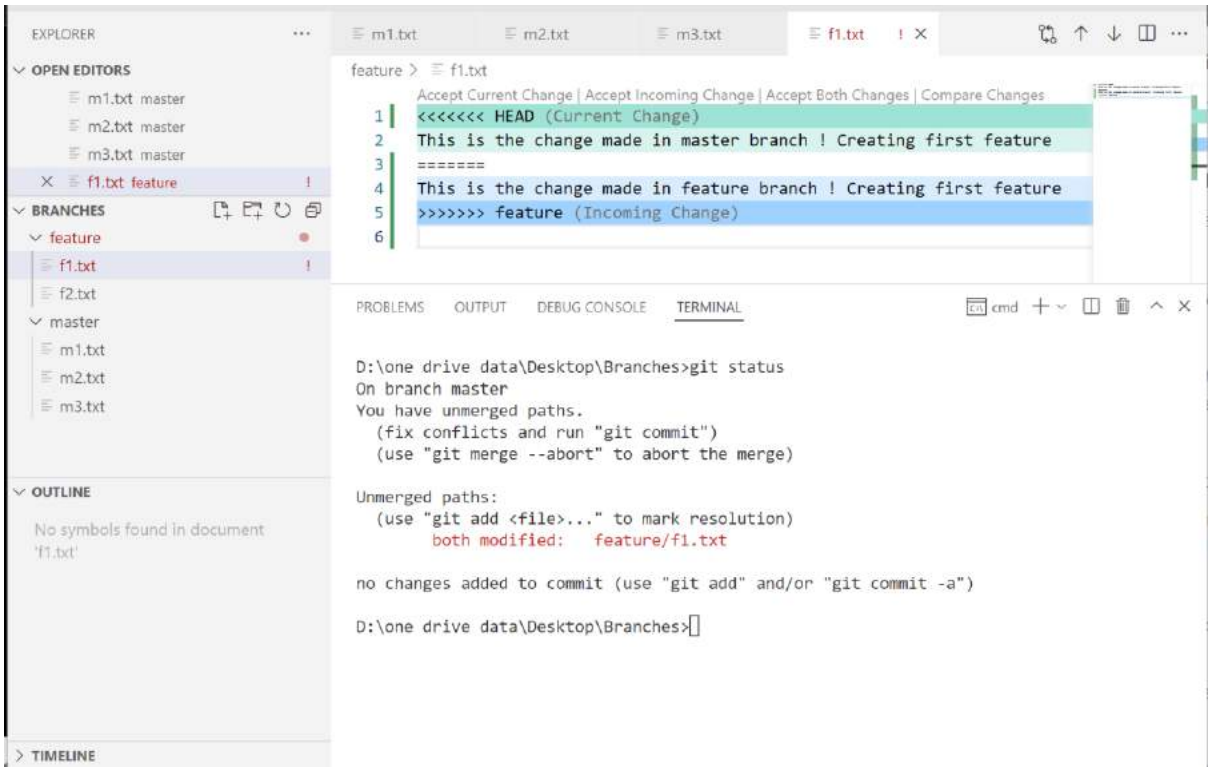


Fig. git status shows how to handle conflicts

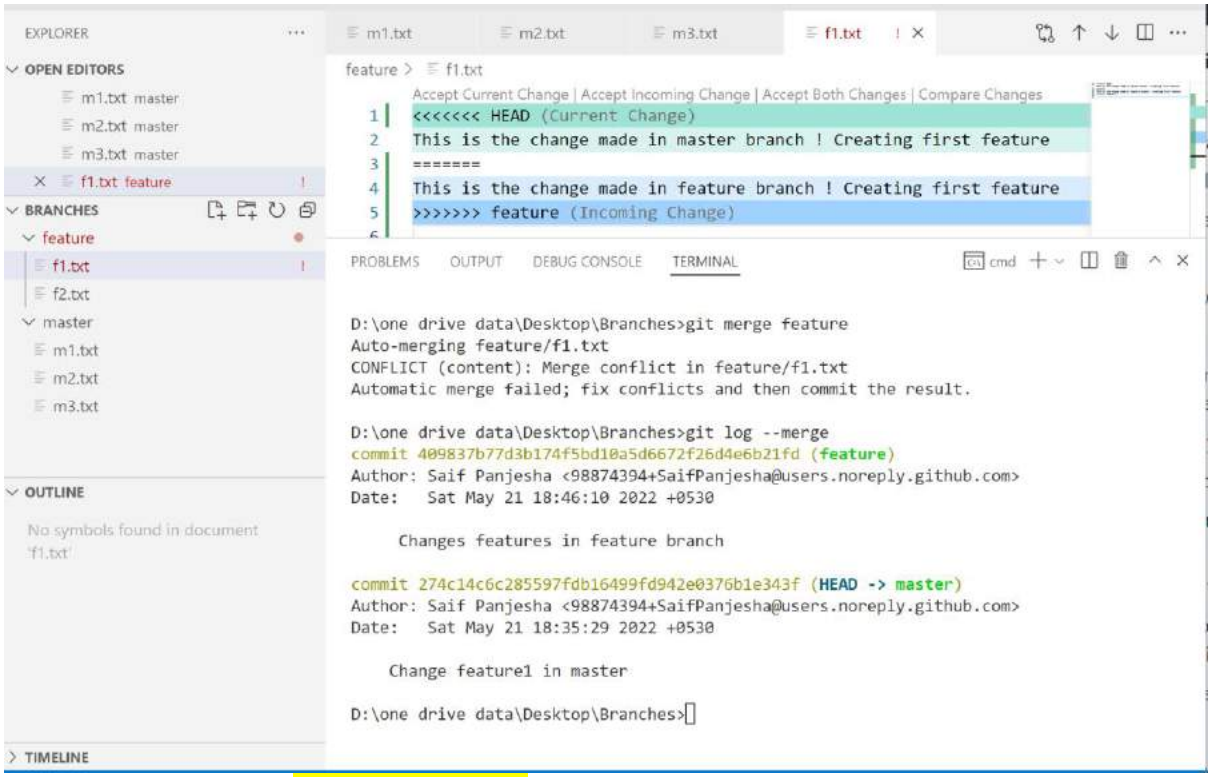


Fig. **git log - -merge** show how to handle conflicts

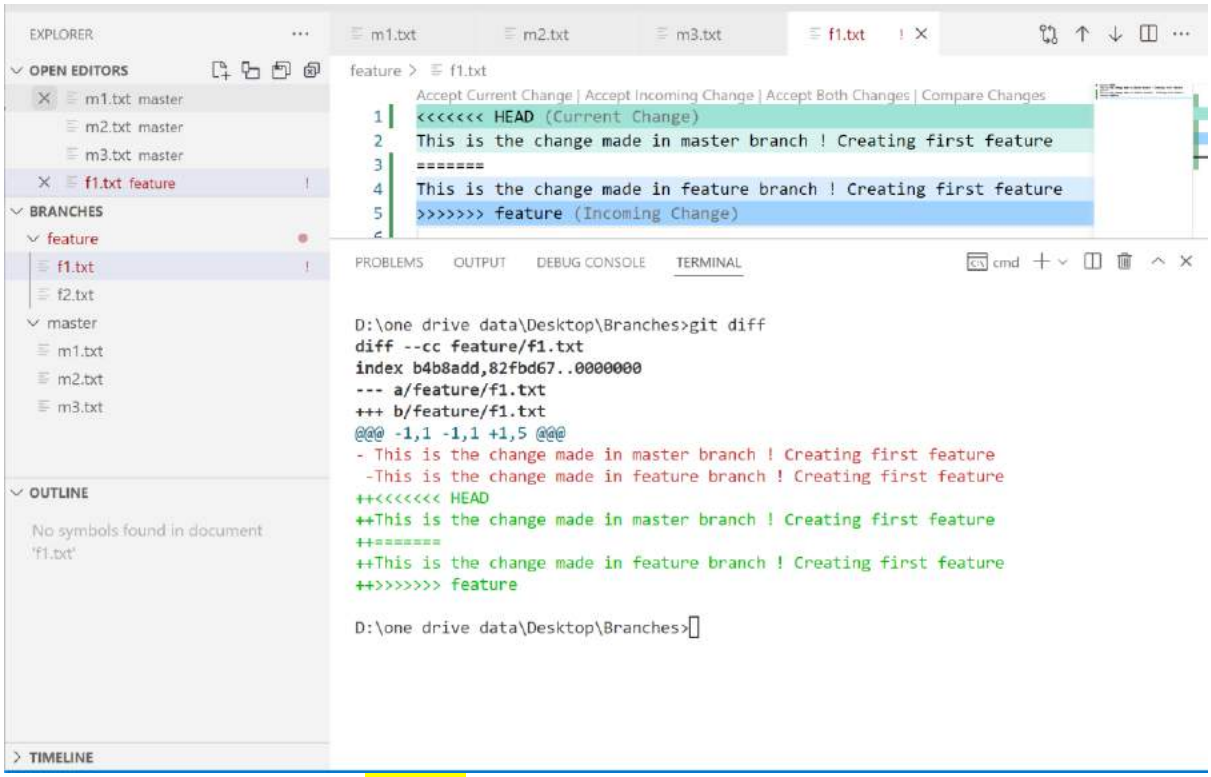


Fig**. git diff** show how to handle conflicts

- **Use git-reset or git merge --abort to cancel a merge that had conflicts**

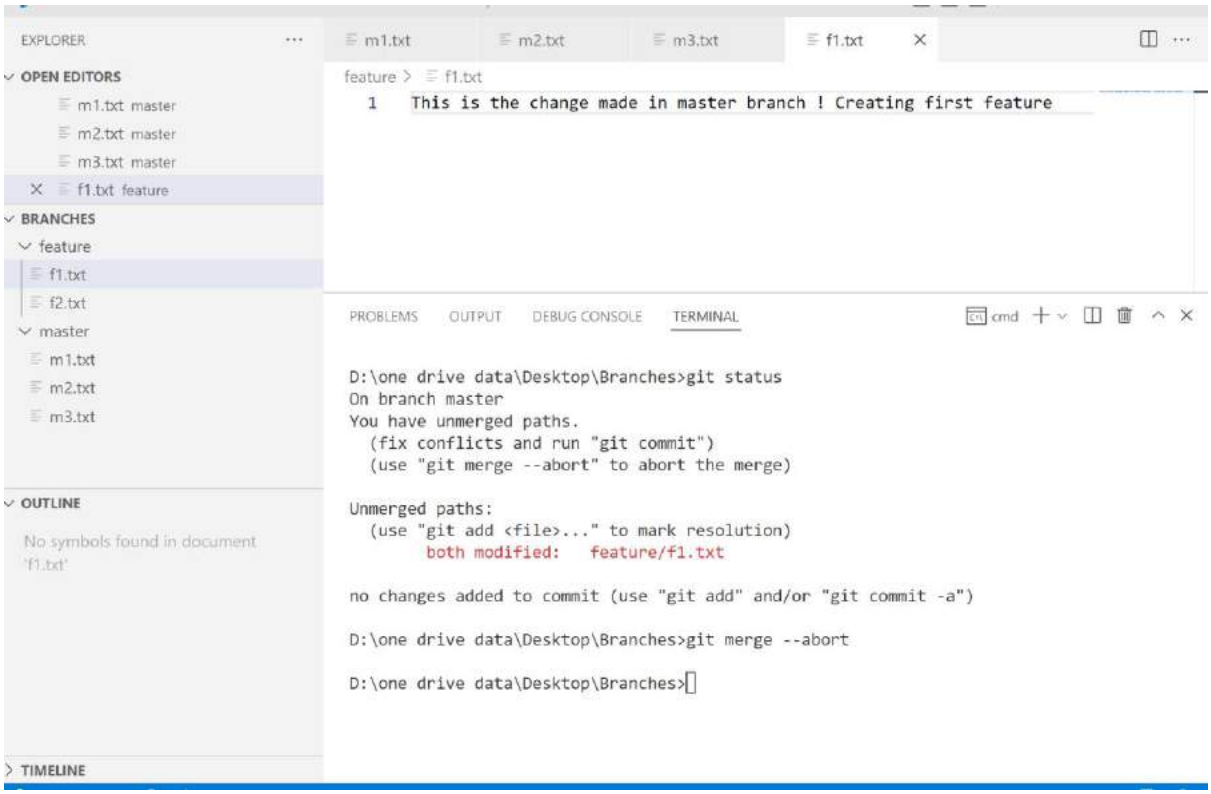


Fig. git merge --abort to cancel a merge that had conflicts

- **Accepting Current Change in VS code and resolving the issue:**

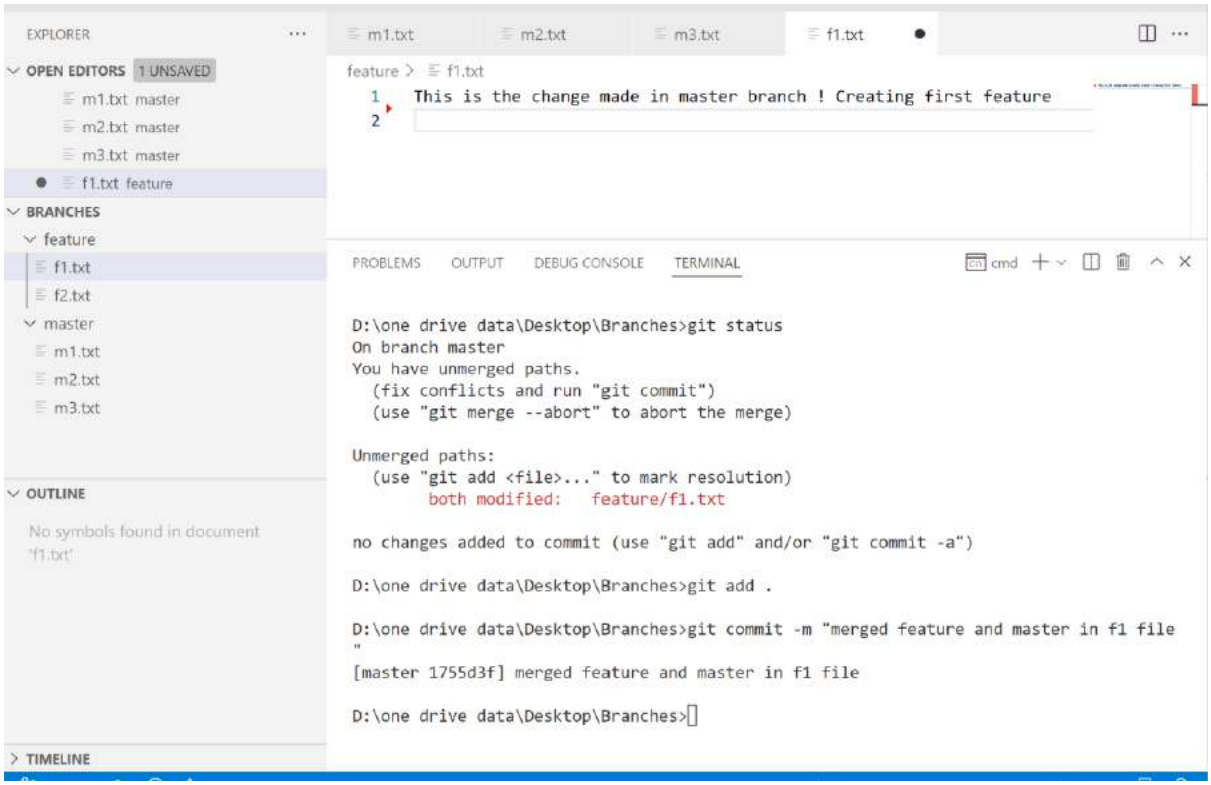


Fig. Merged current changes in master branch

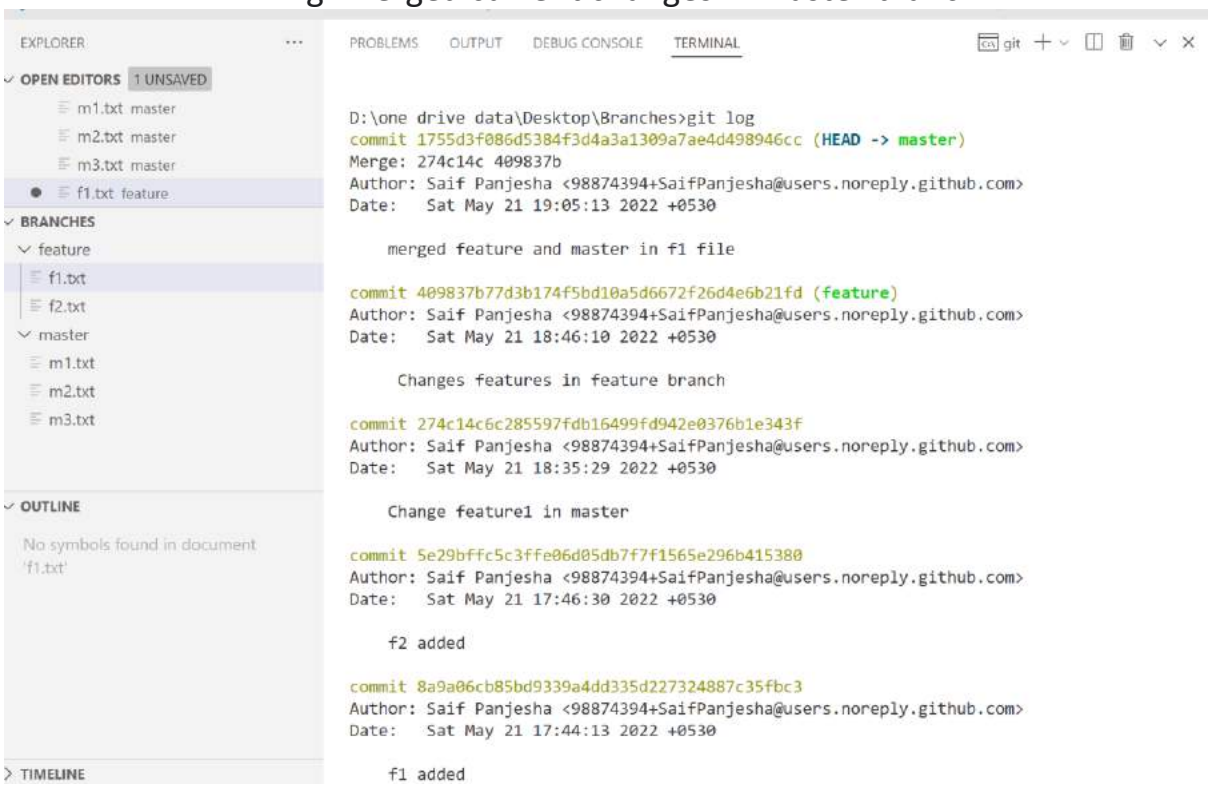


Fig.History committed in master branch

Understanding “git cherry-pick”

The command git *cherry-pick* is typically used to introduce particular commits from one branch within a repository onto a different branch. A common use is to forward- or back-port commits from a maintenance branch to a development branch.

Merge vs Rebase vs Cherry Pick

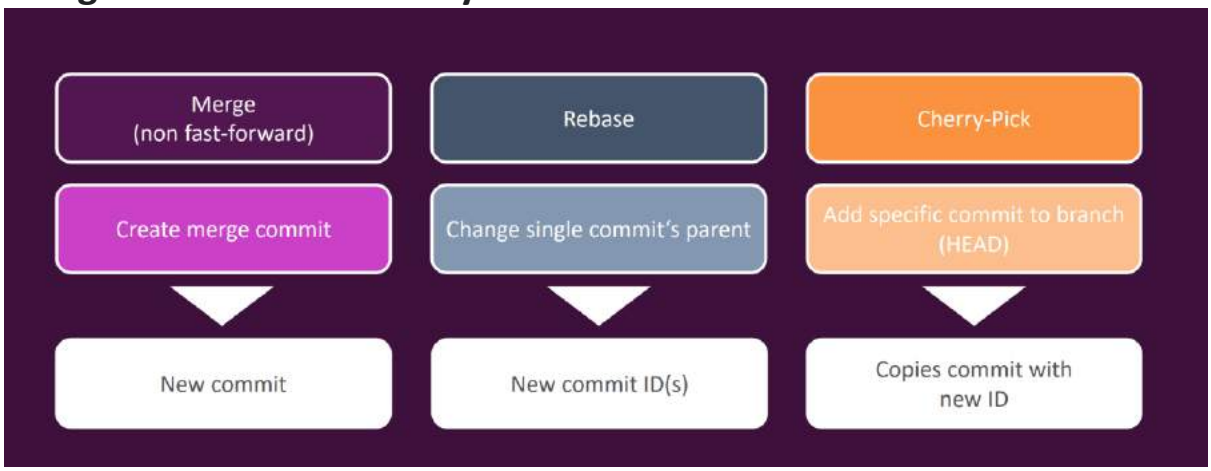


Fig. Quick recap

Scenario: A person is working on two different branches and typo mistake in code then we have added specific commit to branch HEAD.

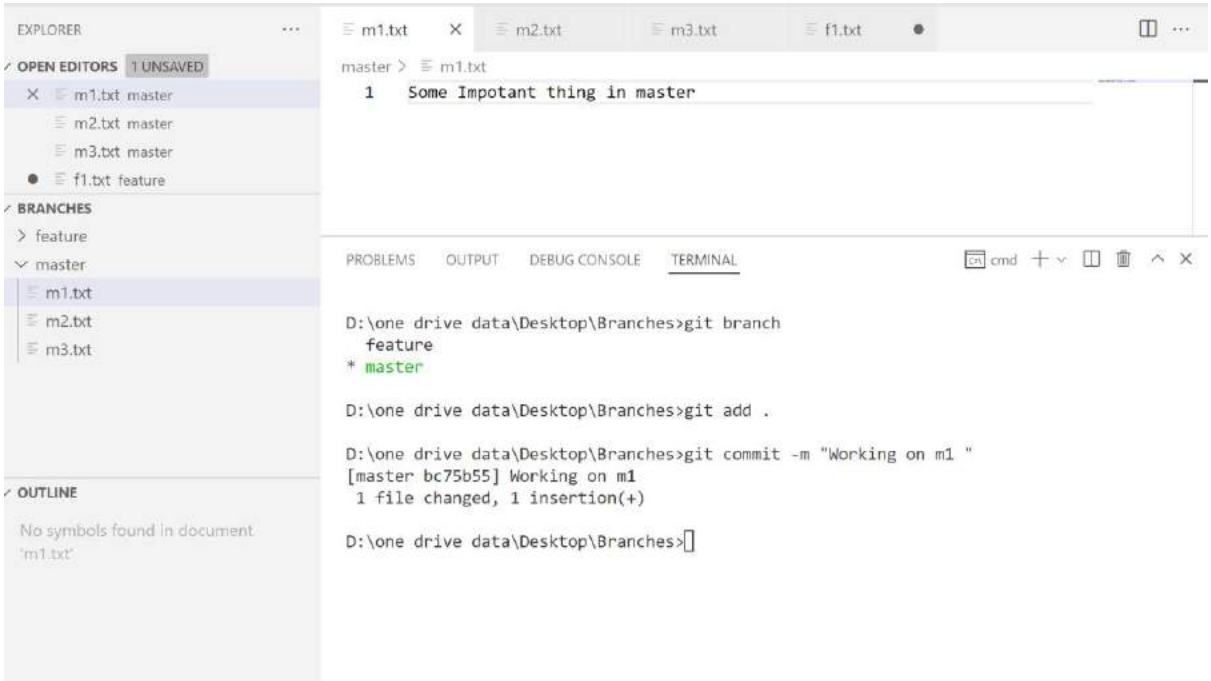


Fig. Working m1 file of master and typo mistake in Important

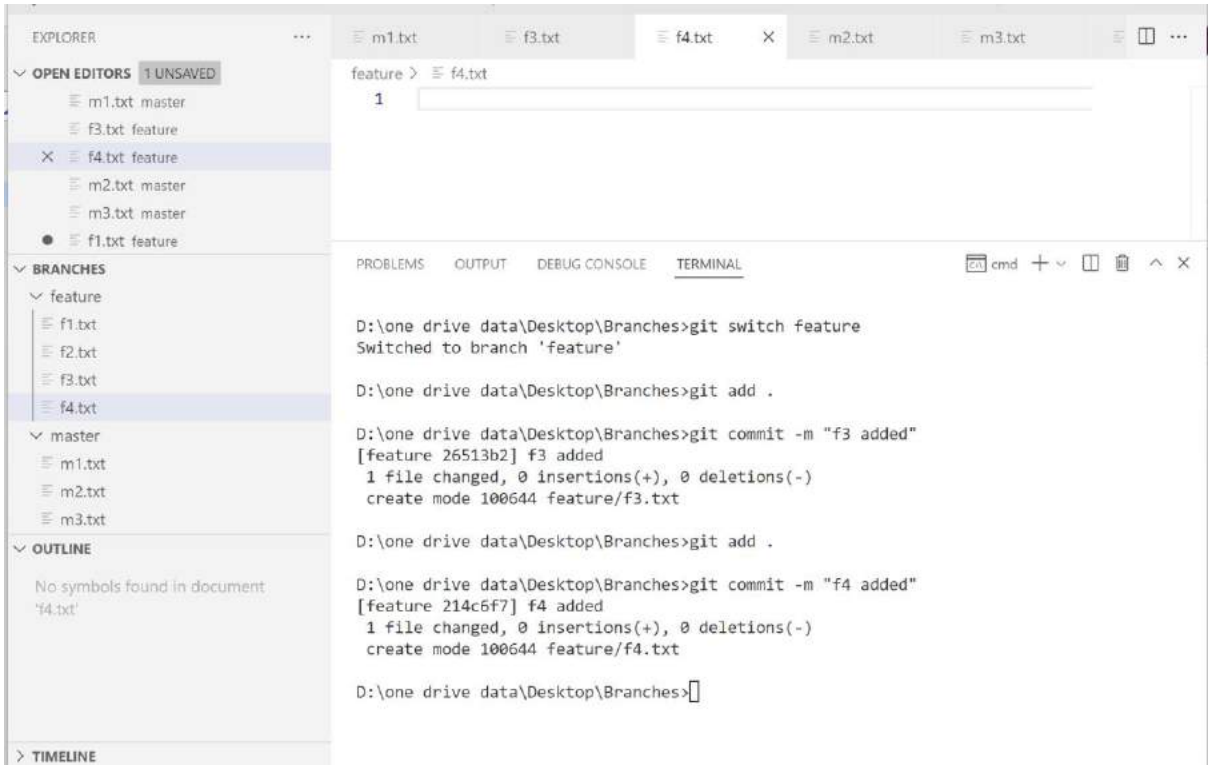


Fig. Building Features f3.txt and f4 .txt in feature Branch

Now Suddenly, we realised their typo in code of m1 file we need to fix it

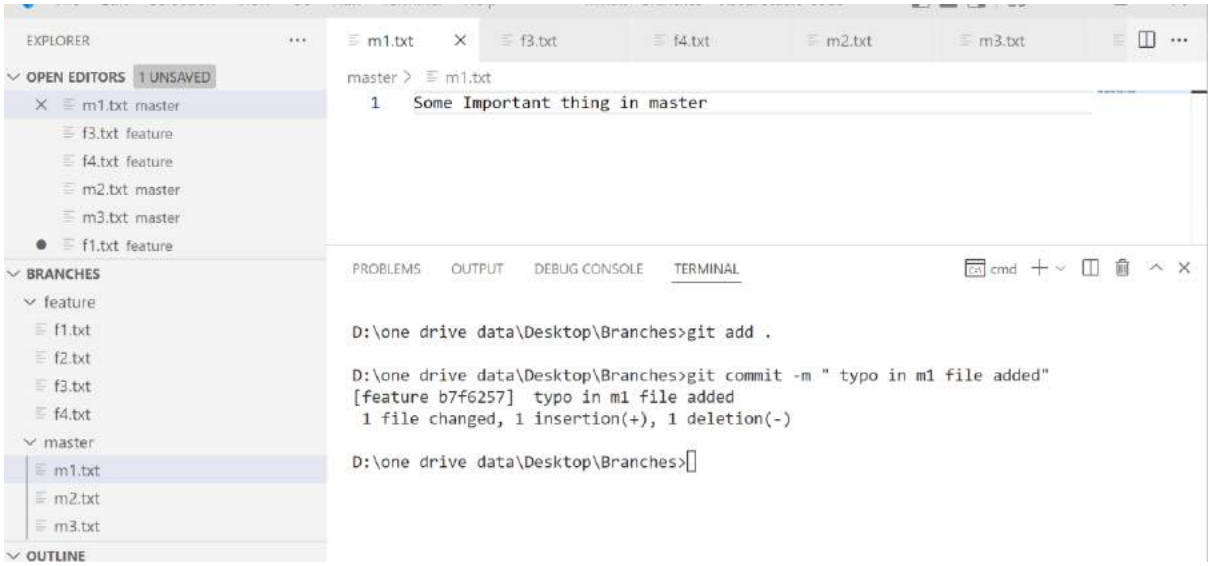


Fig. Shows Fix typo in m1 master file

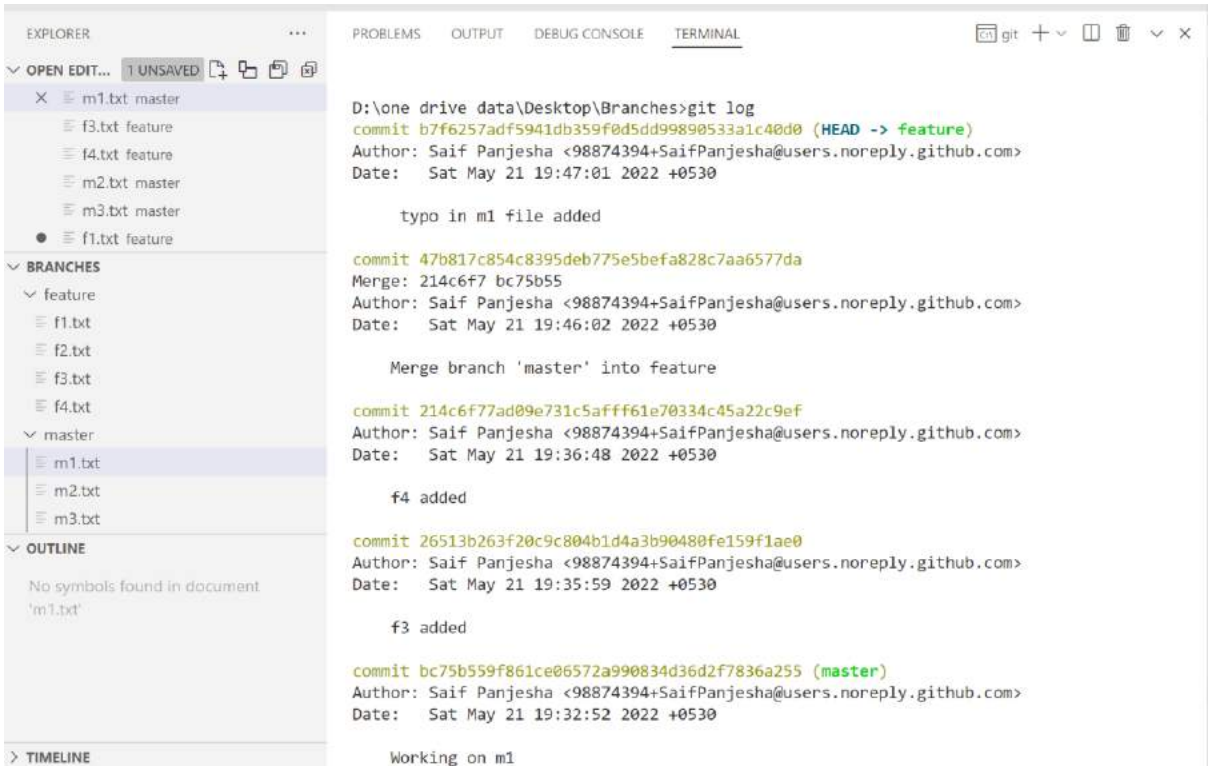


Fig. shows commit history of feature branch

Problem: Now, if we merge feature branch into master then all commits will be going to add. But I want merge the specific typo commit of feature branch into master branch.

To resolve the problem introduced "**git cherry-pick**"

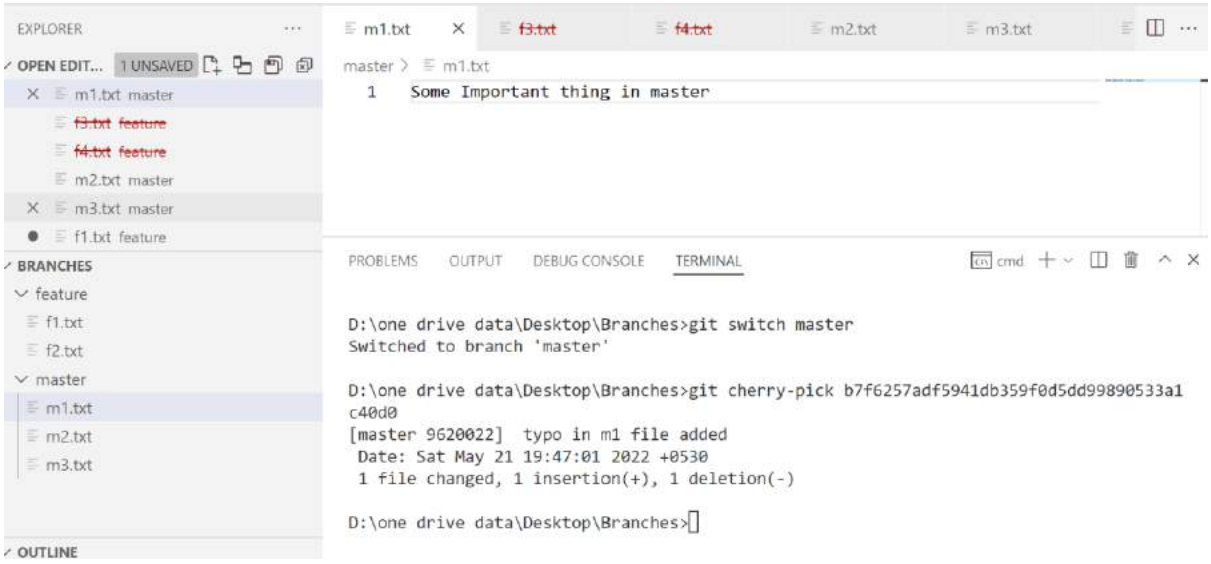


Fig. specific commits has been added to master branch

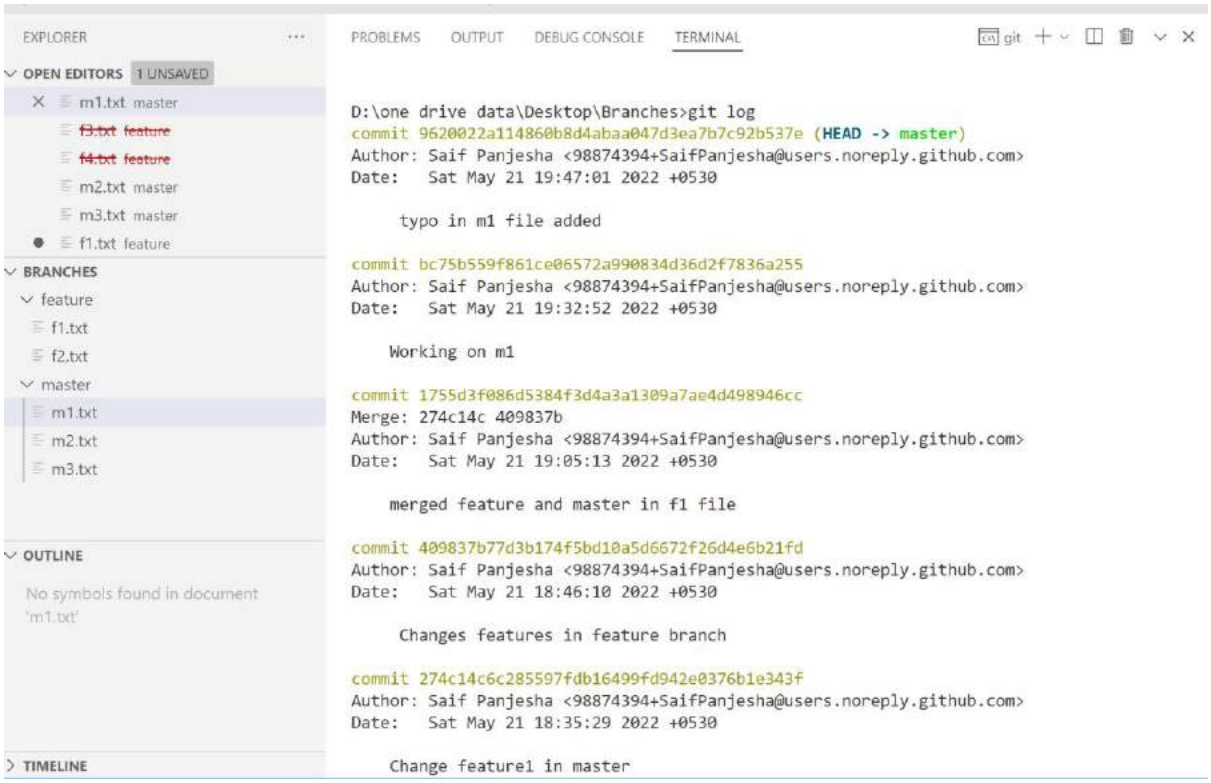


Fig. git log shows that commit is successful.

Master branch:

D:\one drive data\Desktop\Branches>git log
commit 9620022a114860b8d4abaa047d3ea7b7c92b537e (HEAD -> master)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date: Sat May 21 19:47:01 2022 +0530

typo in m1 file added

Feature branch:

commit b7f6257adf5941db359f0d5dd99890533a1c40d0 (HEAD -> feature)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date: Sat May 21 19:47:01 2022 +0530

typo in m1 file added

Clever Note: copies commit is with new ID because of "git cherry-pick"

Working with “git tag”:

- **Tags are ref's that point to specific points in Git history**
- **Git supports two types of tags: lightweight and annotated.**
- **Light weighted: it’s just a pointer to a specific commit.**
- **Annotated: stored as full objects in the Git database**

Lightweight tags:

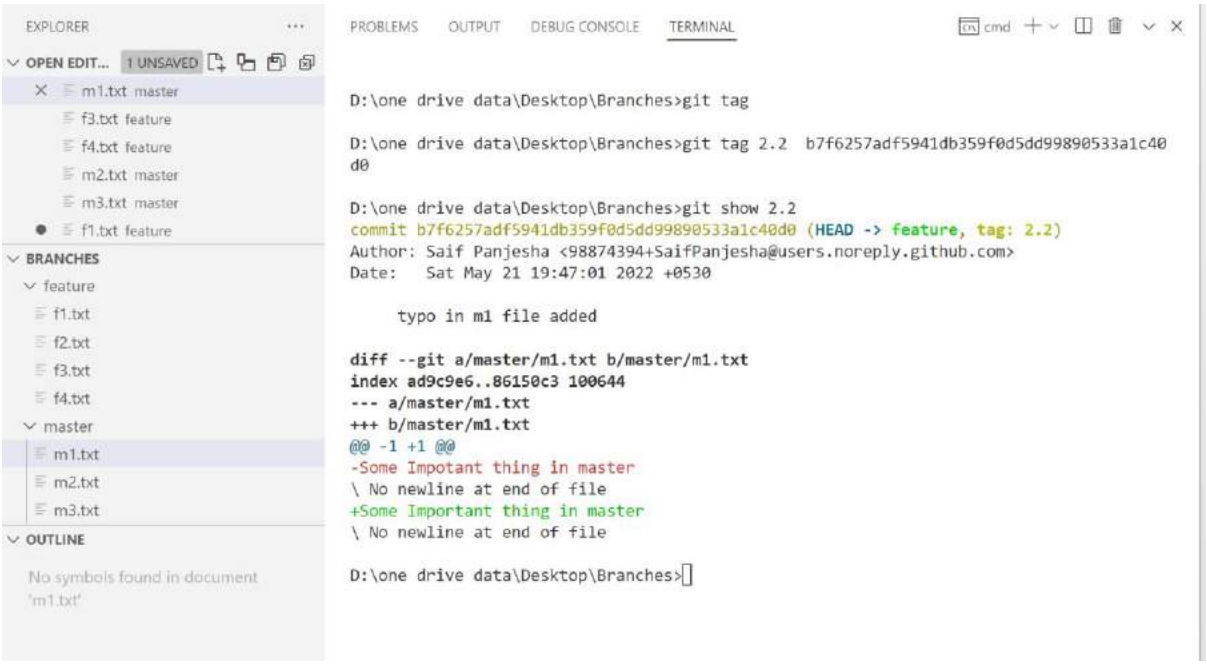


Fig. Lightweight tags Example

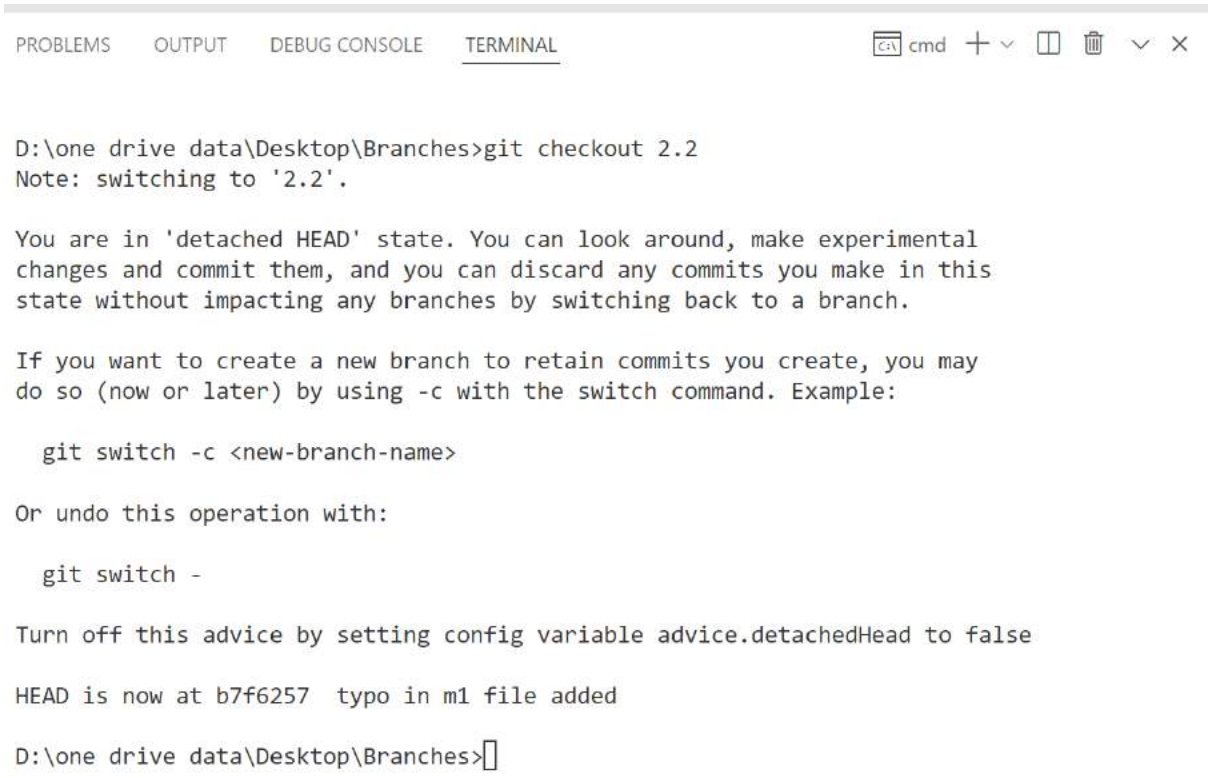


Fig. Shows we can checkout with tags in detached head (lightweight tag)

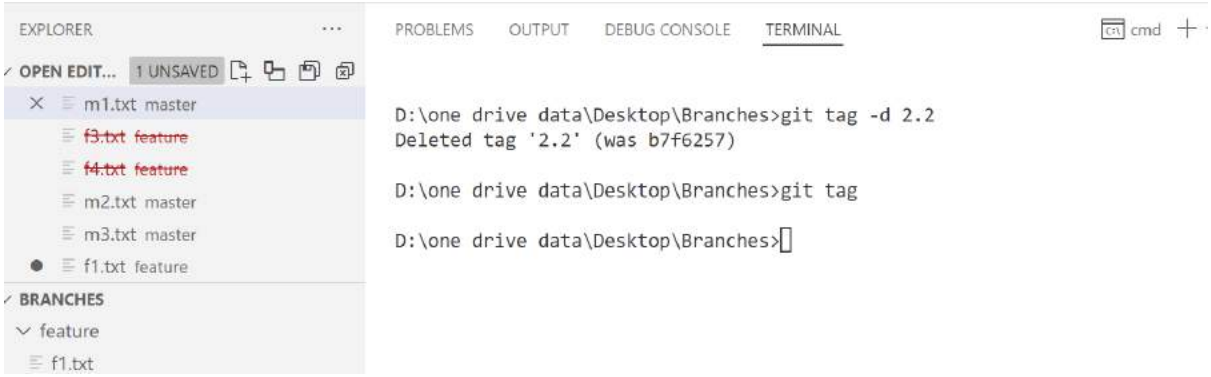


Fig. Successful tag deleted

Annotated tags:

stored as full objects in the Git database

Fig. Annotated tags

Wrap Up Git Diving Deeper:

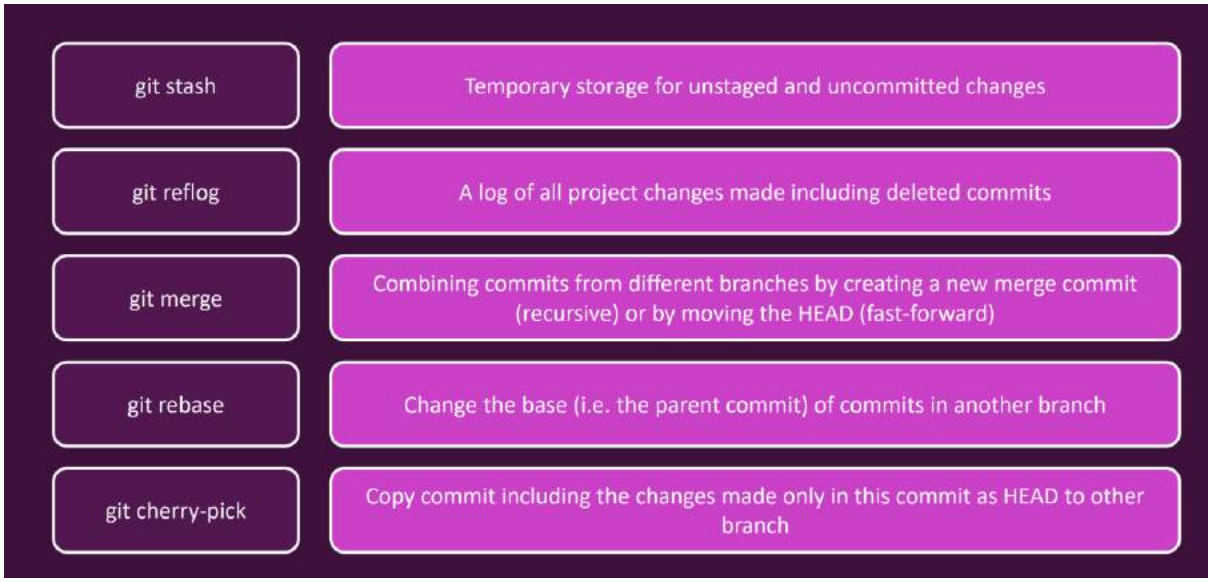


Fig. Git Diving Deeper commands.

From local to remote understanding:

Fig. Git knowledge with GitHub in cloud

Module Overview:

- **What is GitHub? How Git and GitHub are connected?**
- **Remote Branches, Remote Tracking Branches & Local Tracking Branches**
- **Understanding Upstream and Git clone**

What is GitHub?

Fig. shows about Git and GitHub**.**

How Git and GitHub are connected?

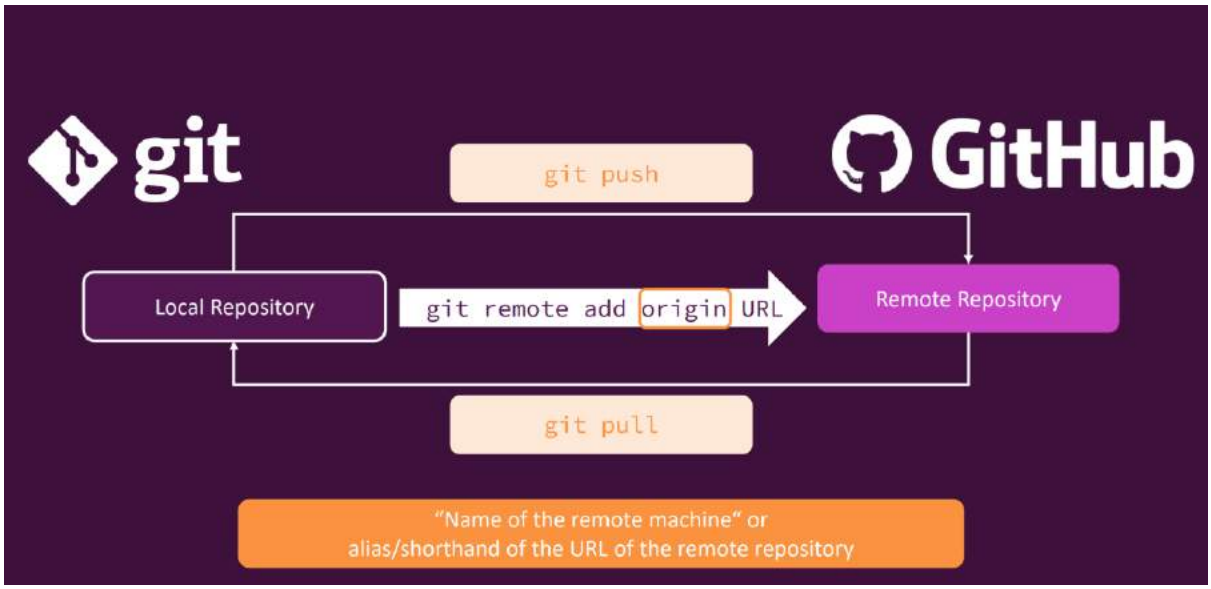


Fig. shows connecting git and GitHub - local to remote repository

Creating a GitHub account and introducing GitHub

1. Open https://github.com in a web browser, and then select **Sign up**.

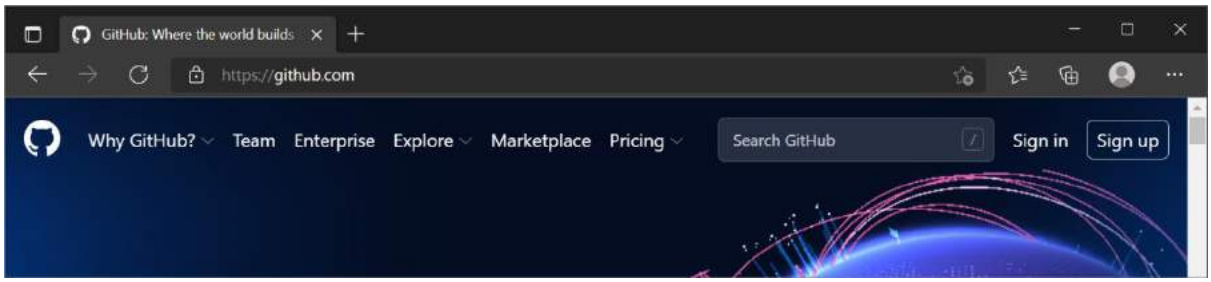


Fig. official site

2. **Enter your email** address.

Fig. Add your email address

3. **Create a password** for your new GitHub account, and **Enter a username**, too. Next, choose whether you want to receive updates and announcements via email, and then select **Continue**.

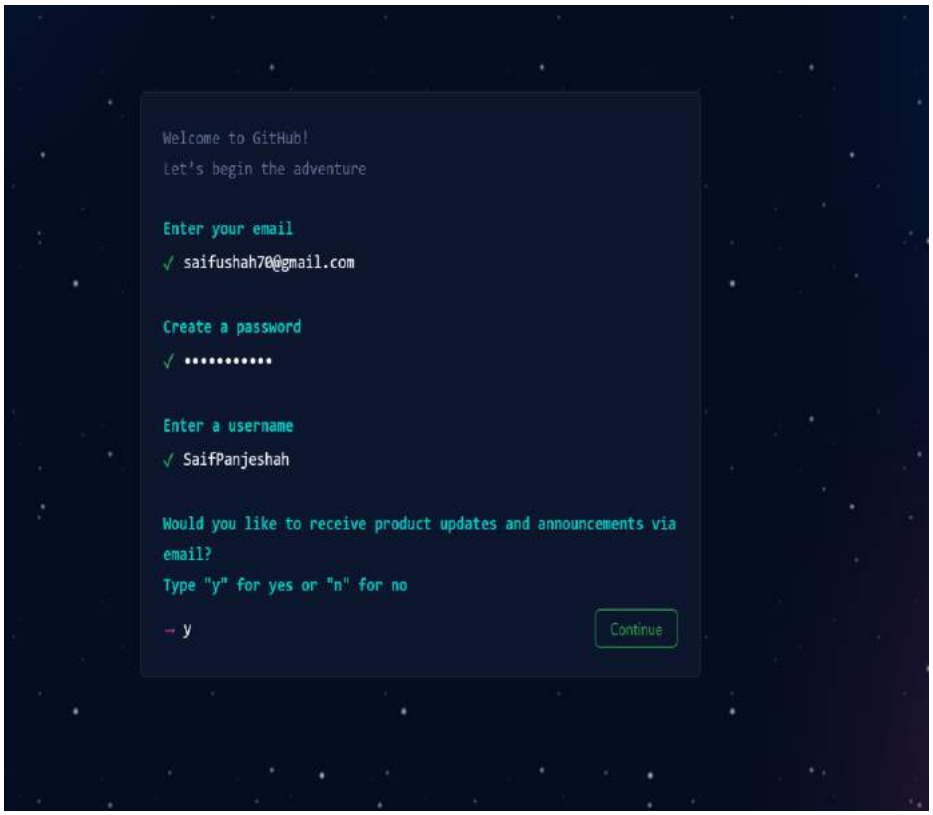


Fig. Creation of Username & Password

4. **Verify your account** by solving a puzzle. Select the **Start Puzzle** button to do so, and then follow the prompts.

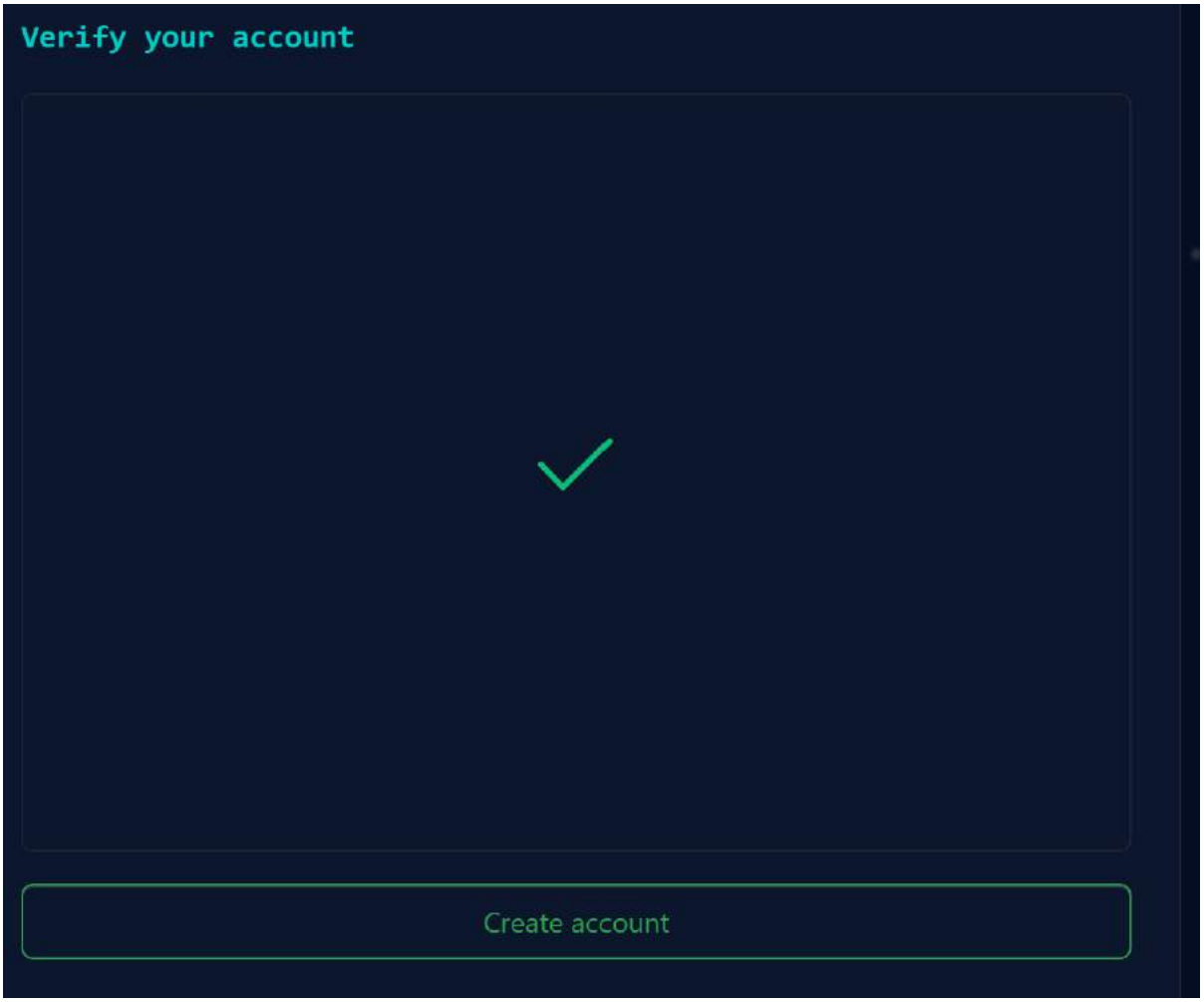


Fig. Account Verified

5. After you verify your account, select the **Create account** button.

6. Next, GitHub sends a launch code to your email address. Type that launch code in the **Enter code** dialog, and then press **Enter**.

Fig. Enter the Code

7. GitHub asks you some questions to help tailor your experience. Choose the answers that apply to you in the following dialogs:
    - **How many team members will be working with you?**
    - **What specific features are you interested in using?**

8. On the **Where teams collaborate and ship** screen, you can choose whether you want to use the Free account or the Team account. To choose the **Free** account, select the **Skip personalization** button.

    Tip

    You can always upgrade your account later. See the **Types of GitHub accounts** page to learn more.

    GitHub opens a personalized page in your browser.

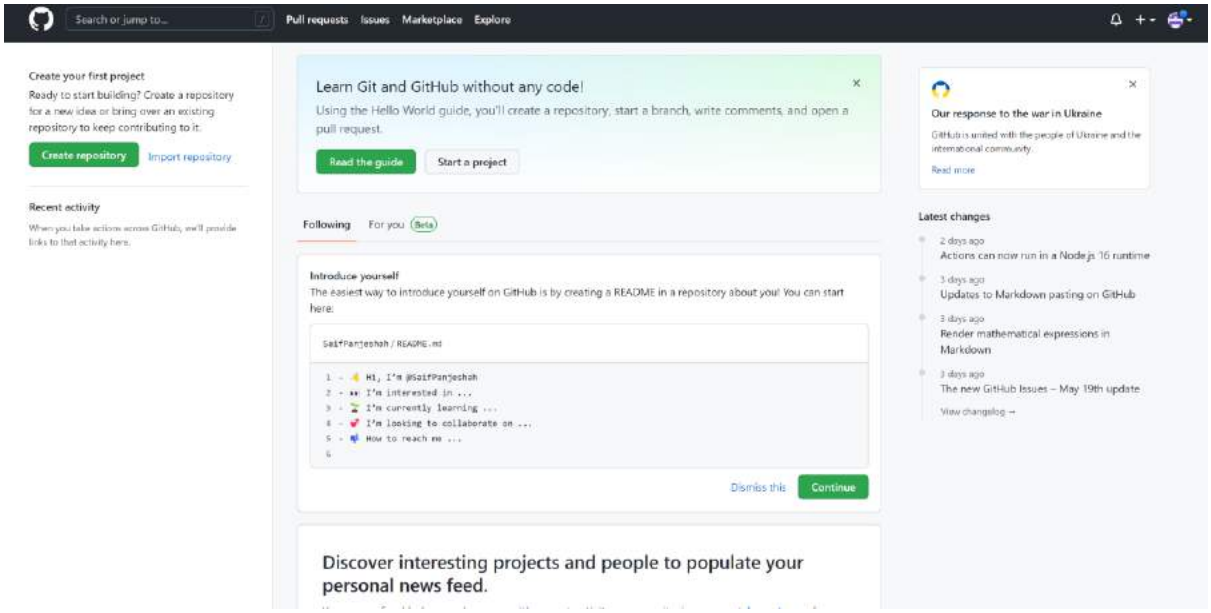


Fig. Account Creation Successful

Creating a repository:

1. Click on create repository on top left corner in account or go to your profile and create repositories in account

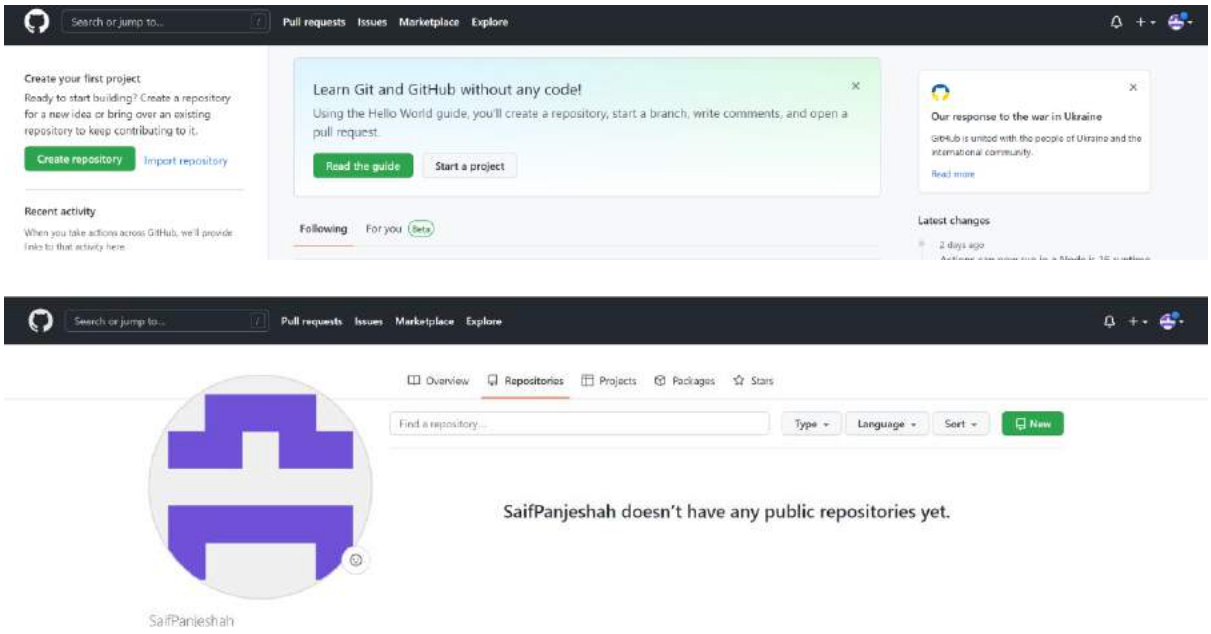


Fig. Creating repository

2. Add your repository name and choose your access public and private Initialize your repository with README file , .gitignore not to choose track from list and choose license others Do's and not Do's .

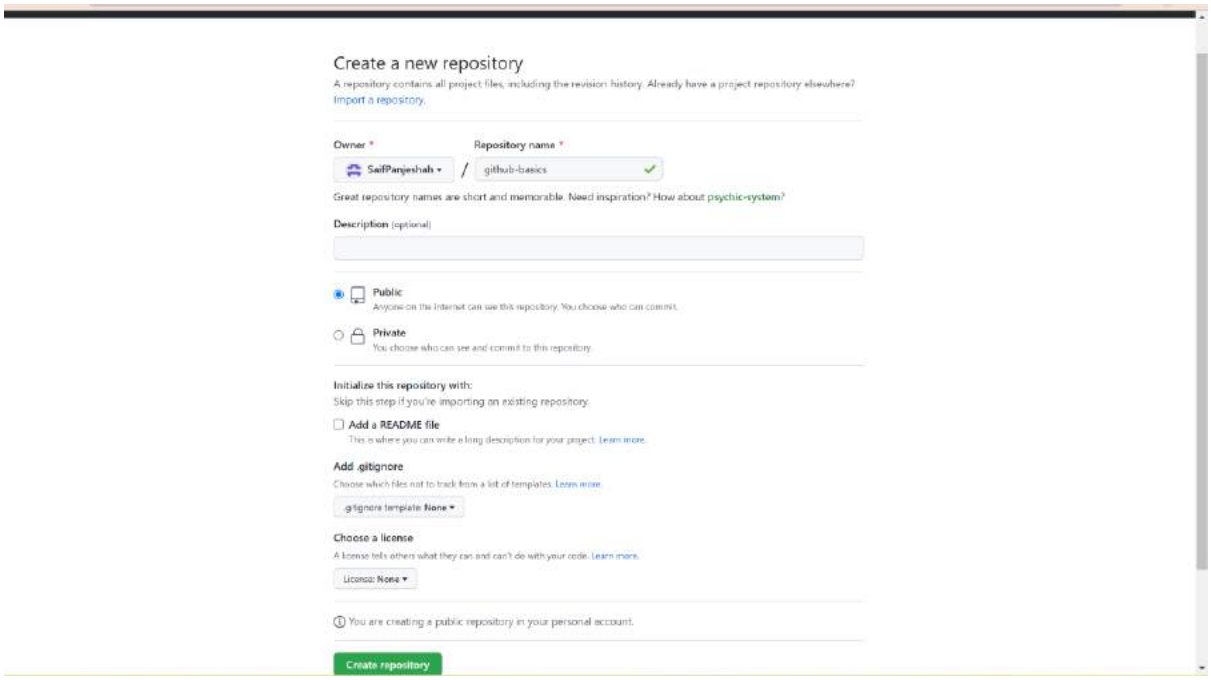


Fig. Created a new repository

3. Quick Step-up to connect with your git account.

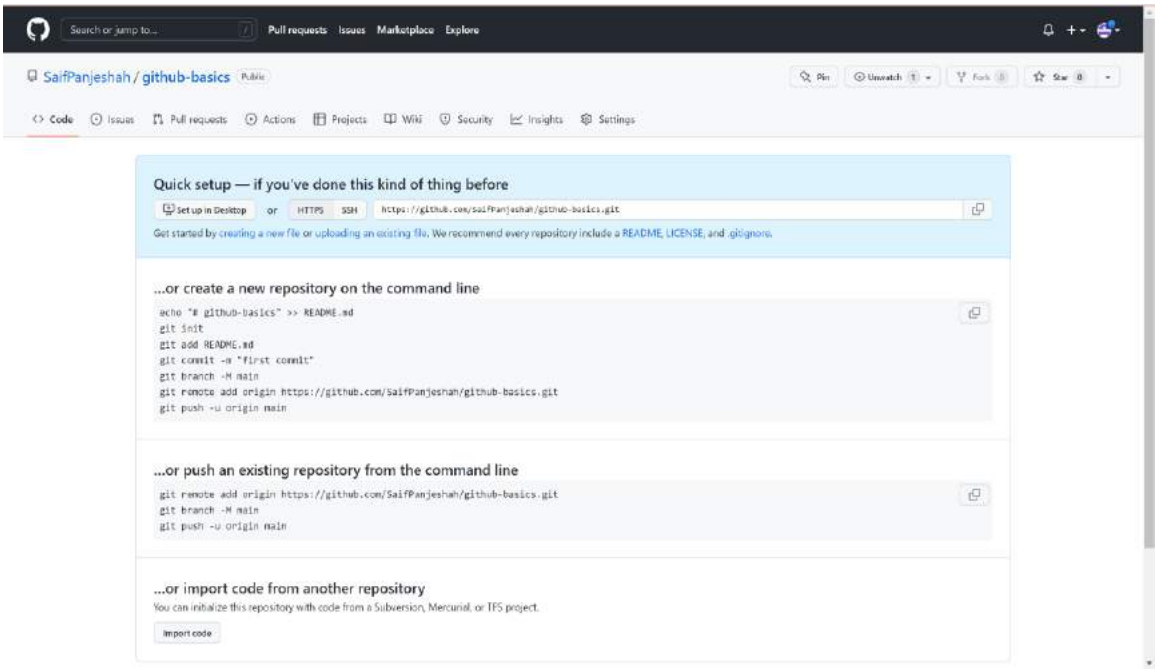


Fig. Setup git on GitHub

4. Click on GitHub Home tab your repository has successful created

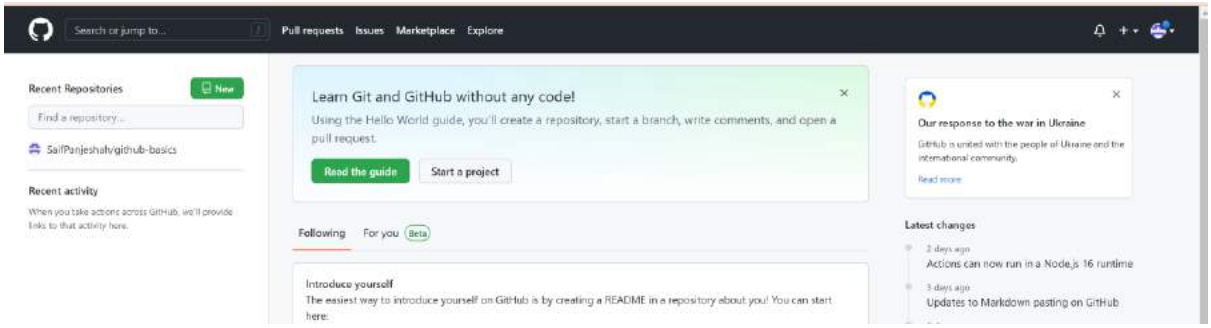


Fig. Repository Created

5. Go to profile and check successful created as popular repository and contribution in the last year

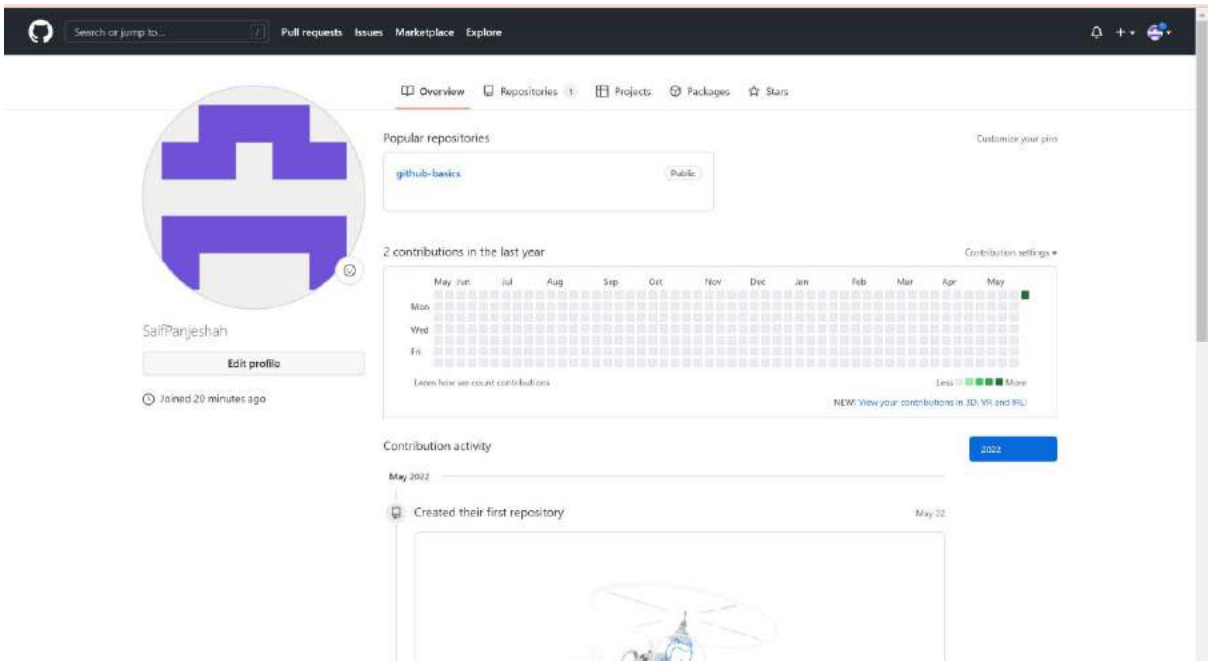


Fig. Successful remote repository added

Connecting Local and remote repository:

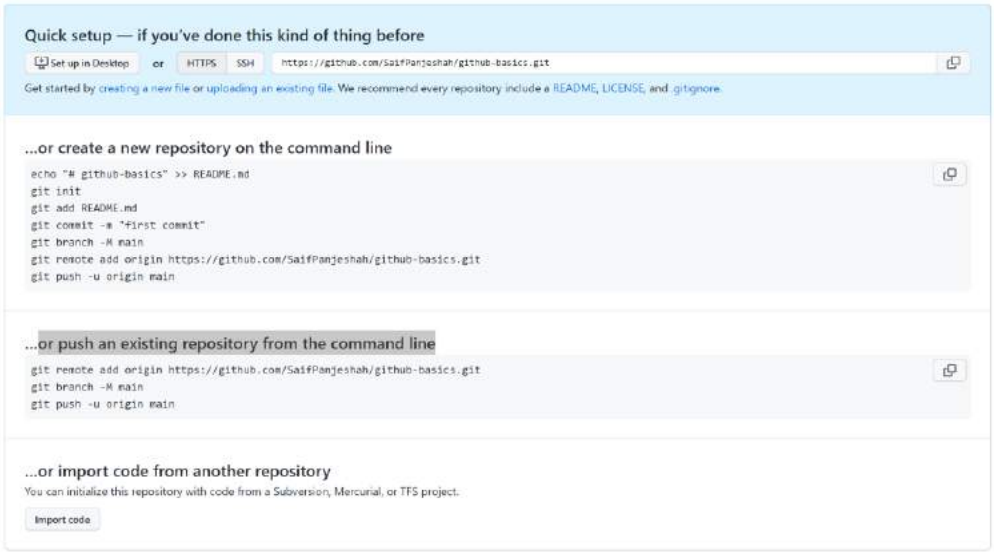


Fig. pushing repository from git

Git local repository

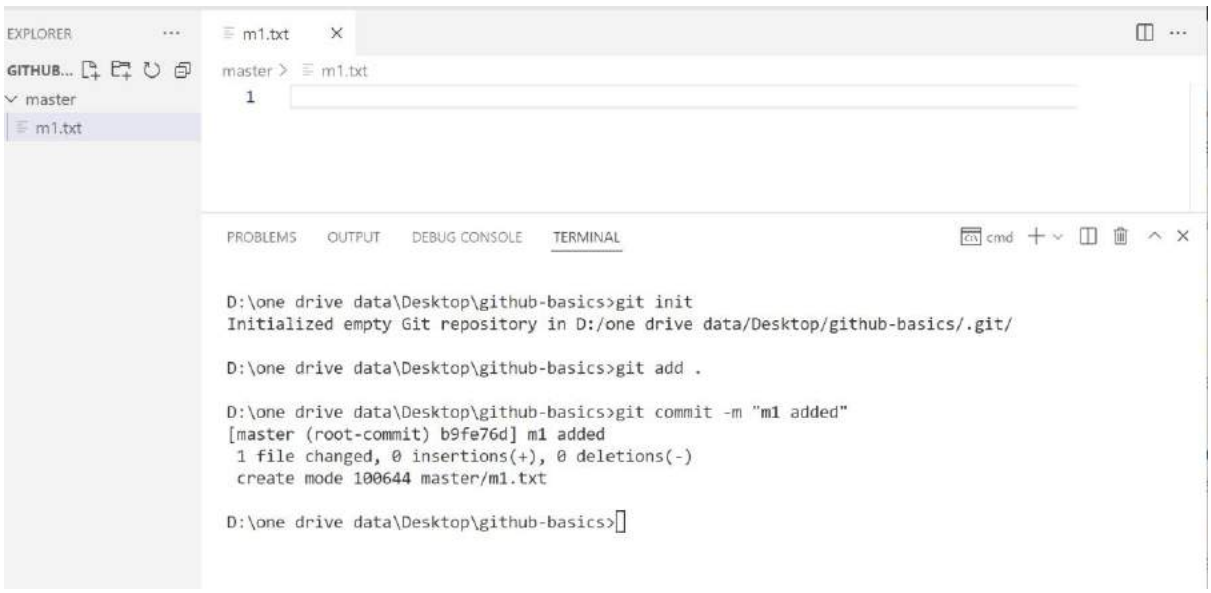


Fig. Creation of git repository

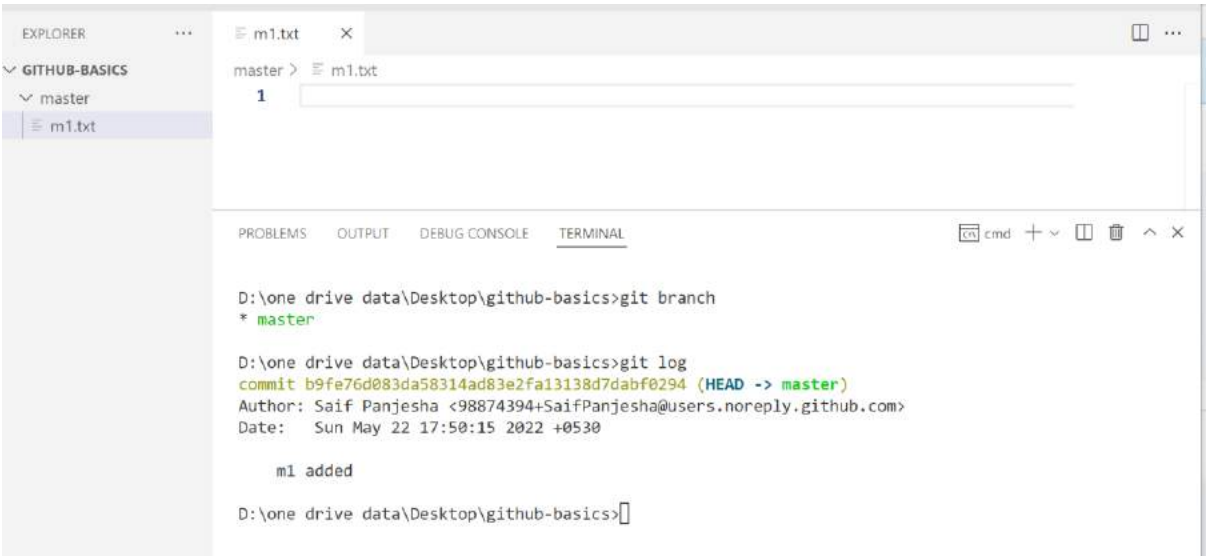


Fig. branches and commit history.

Now, how we can move this local repository to Cloud?

Using Command:

git remote add origin https://github.com/SaifPanjeshah/gitHub1-basics.git
//add the remote connection from local repository

git push origin master **//Bring our local changes, our local information on remote repository**

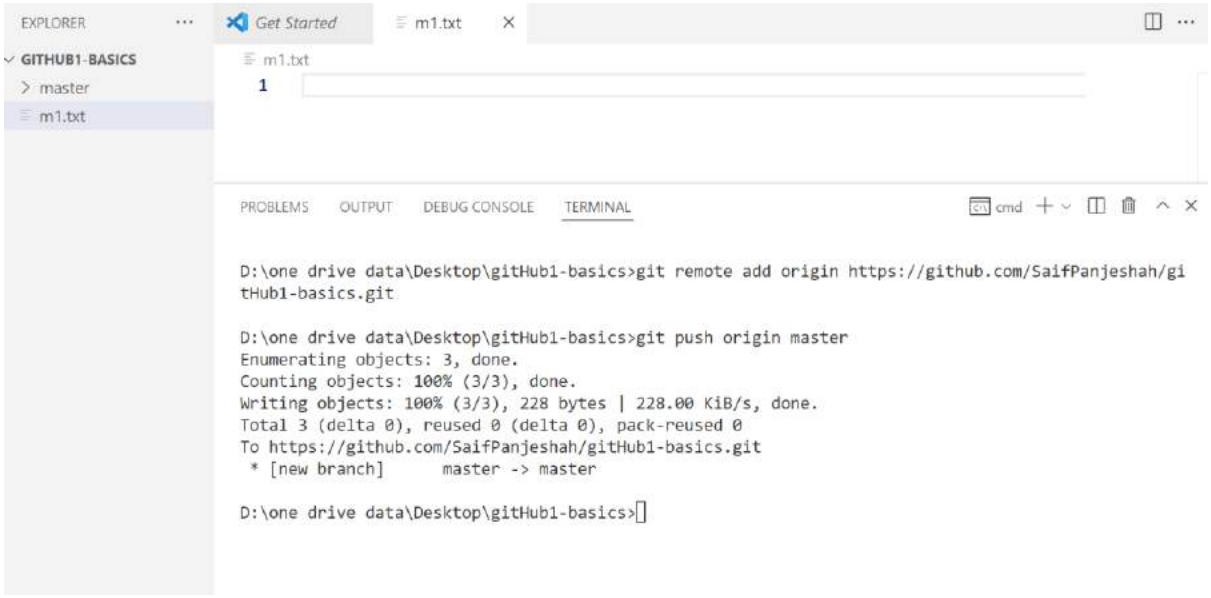


Fig. Successful push our local repository on GitHub

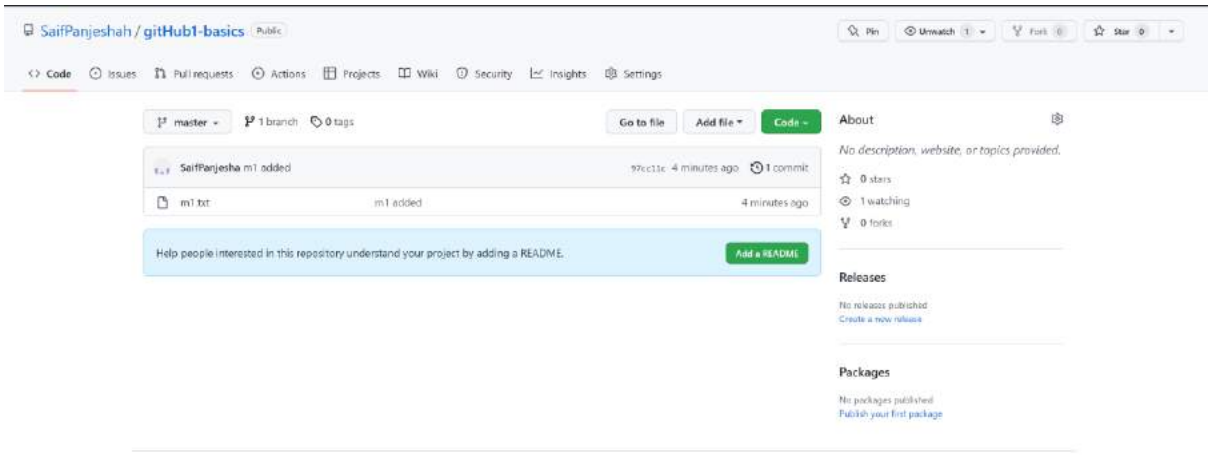


Fig. Successful push our local repository on GitHub via accessing through Internet browser.

Now, how can we push our code with latest approach? Personal Access Token?

Understanding the Personal Access Token:

Creating a token:

a. Verify your email address, if it hasn't been verified yet.

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

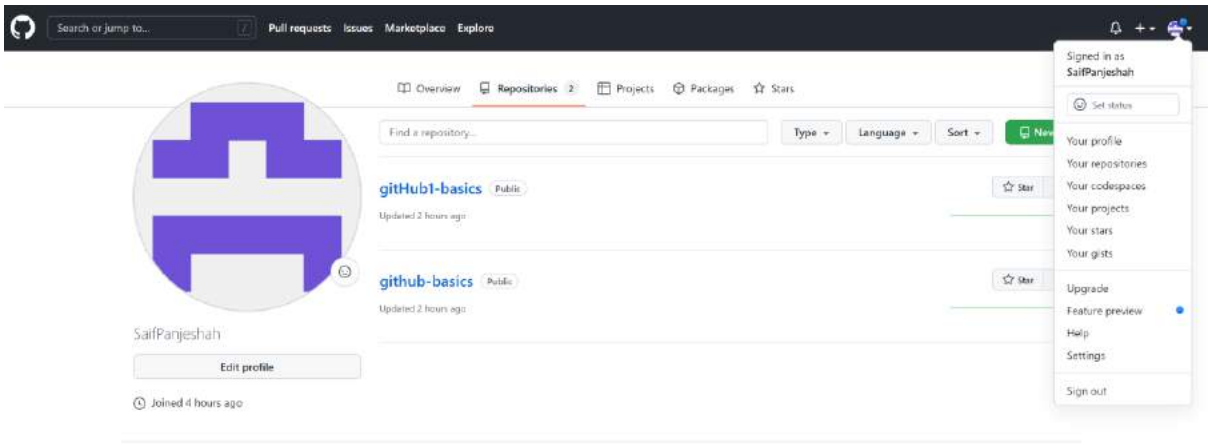


Fig. After verifying email address GitHub profile

c. In the left sidebar, click **Developer settings**

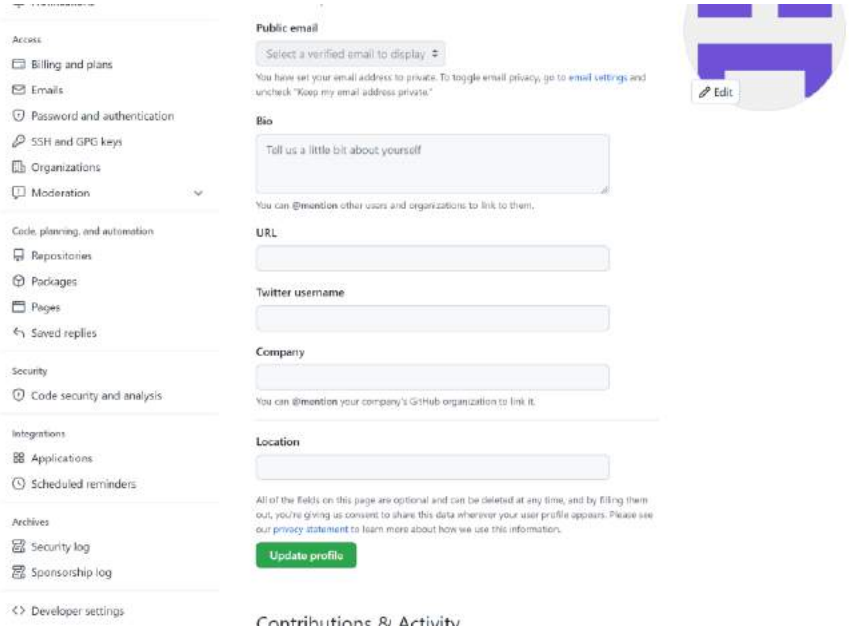


Fig. Setting to open developer setting

d. In the left sidebar, click **Personal access tokens**.

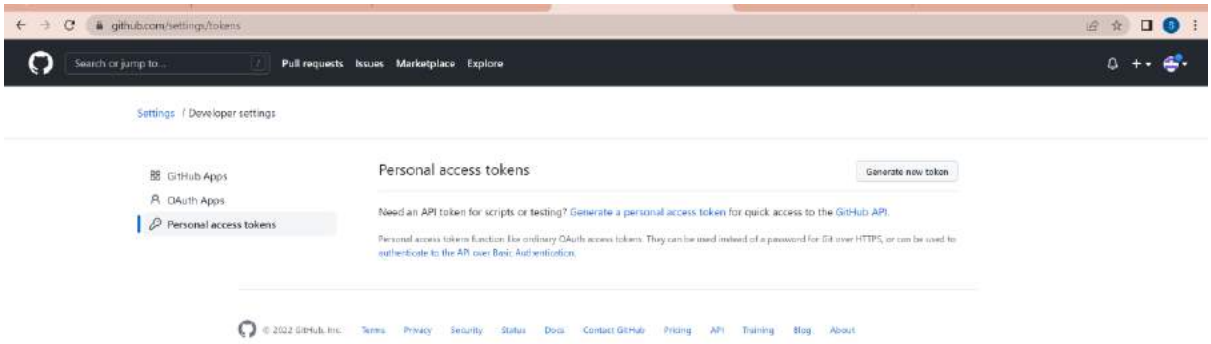


Fig. Personal access tokens

e. Give your token a descriptive name.

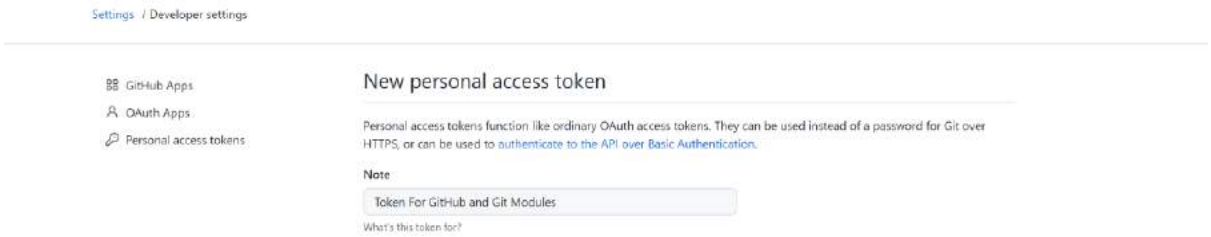


Fig. Descriptive Name of Token

f. To give your token an expiration, select the **Expiration** drop-down menu, then click a default or use the calendar picker.

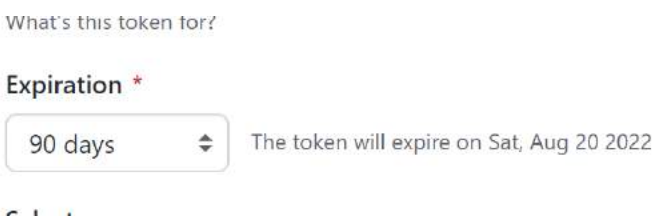


Fig. Expiration days

g. Select the scopes, or permissions, you'd like to grant this token. To use your token to access repositories from the command line, select **repo**.

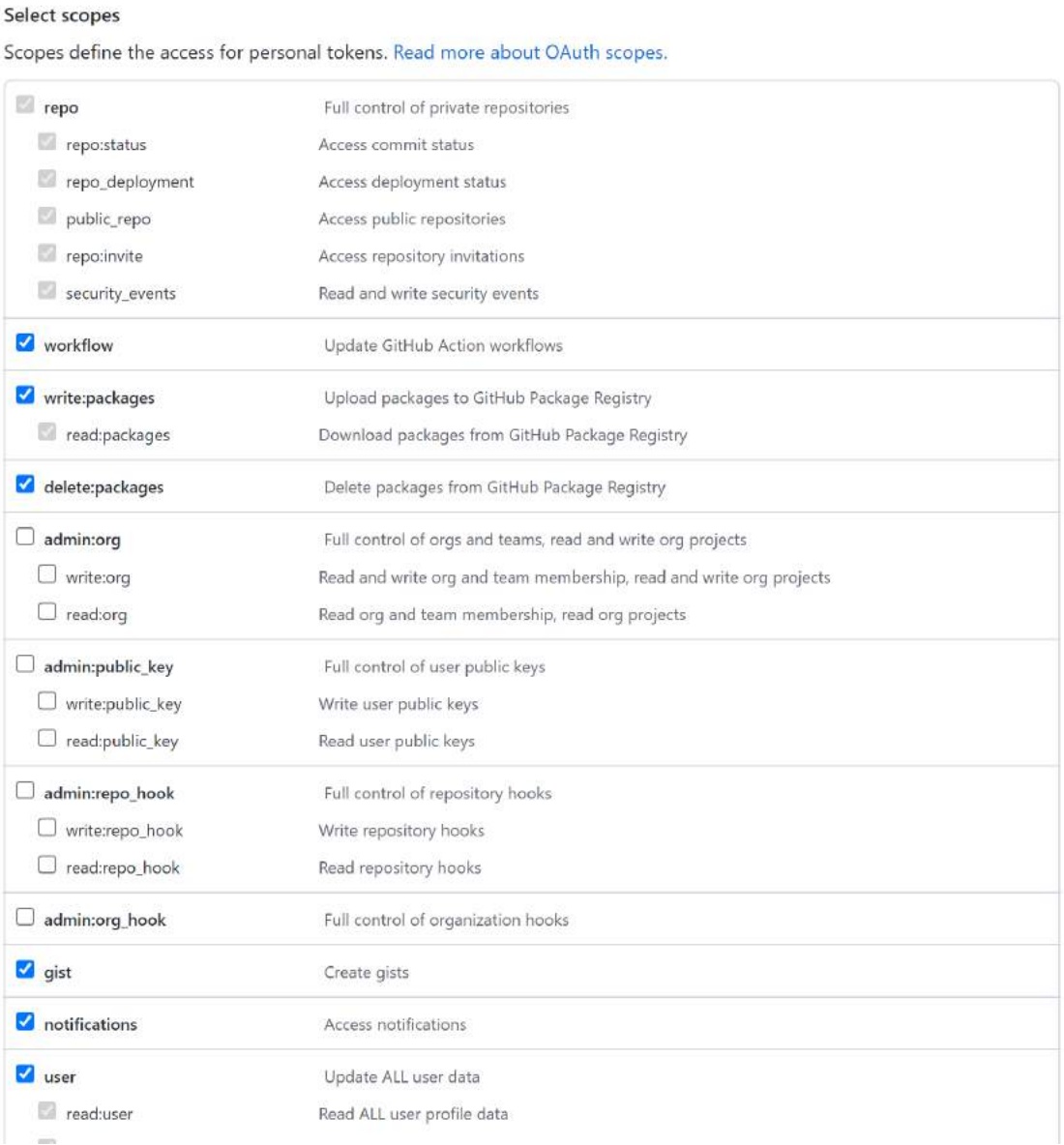


Fig. Permission for tokens excepts admin

h. Click **Generate token**.

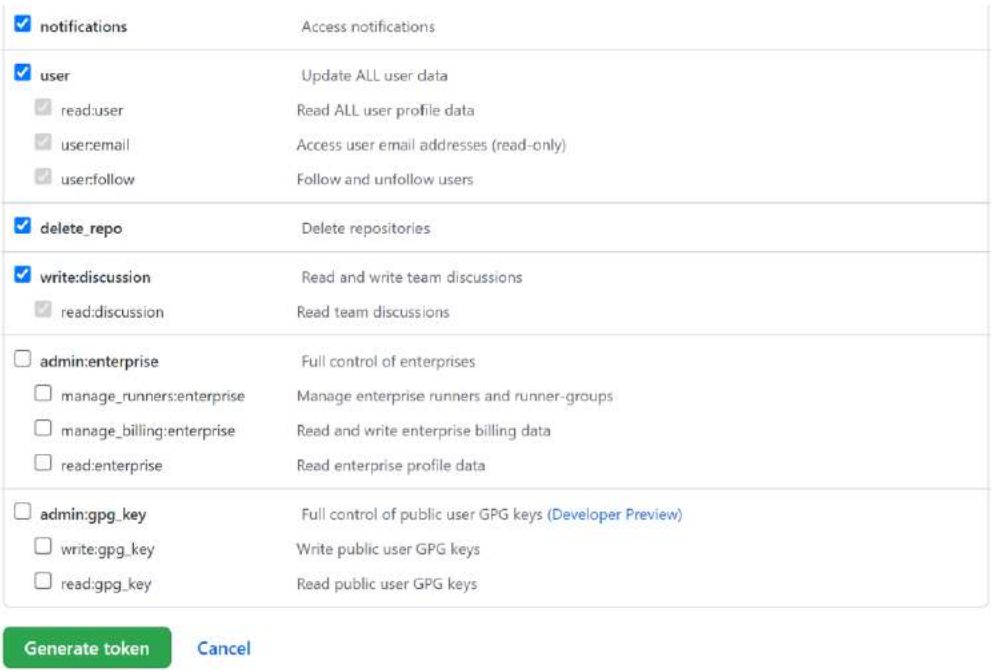


Fig. Generating Token

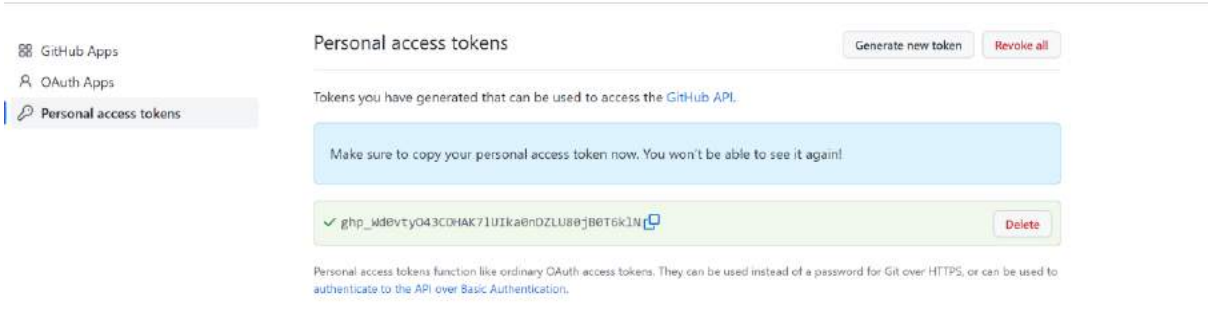


Fig. Token Successful Generated

Pushing a Second Commit

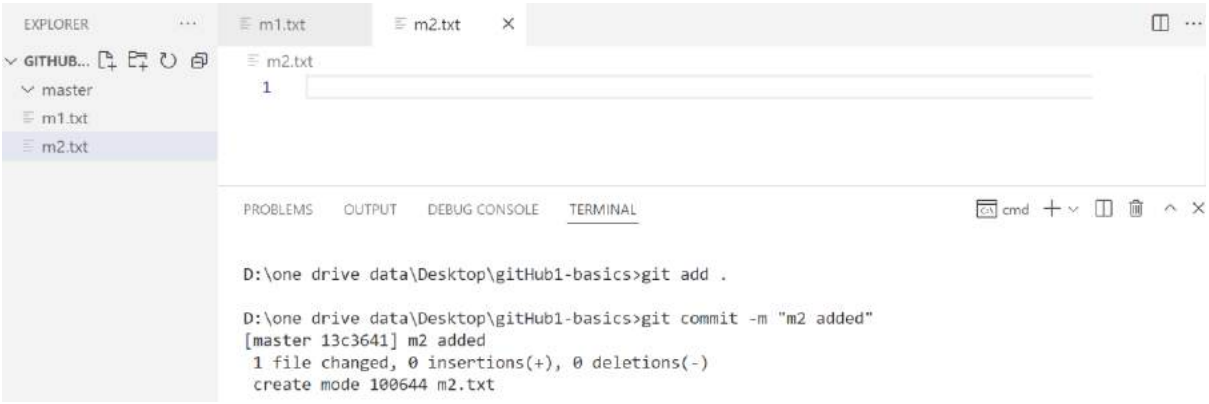


Fig. Created M2 file

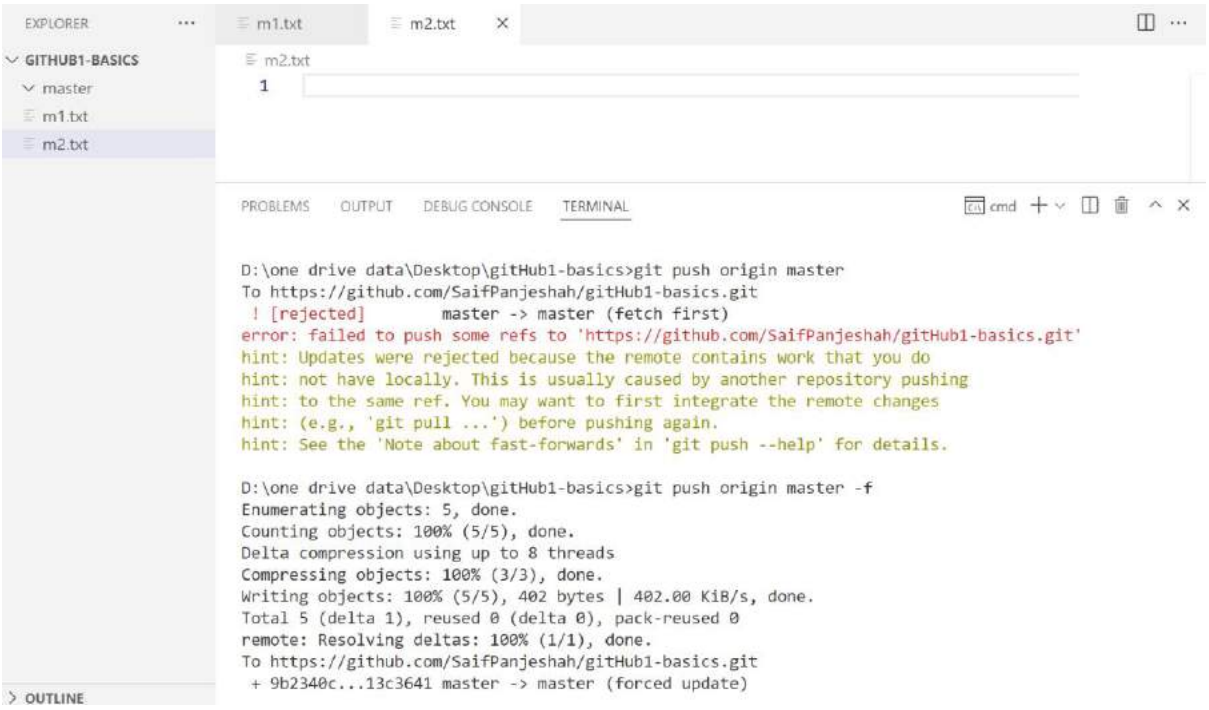


Fig. Push M2 file to remote repository

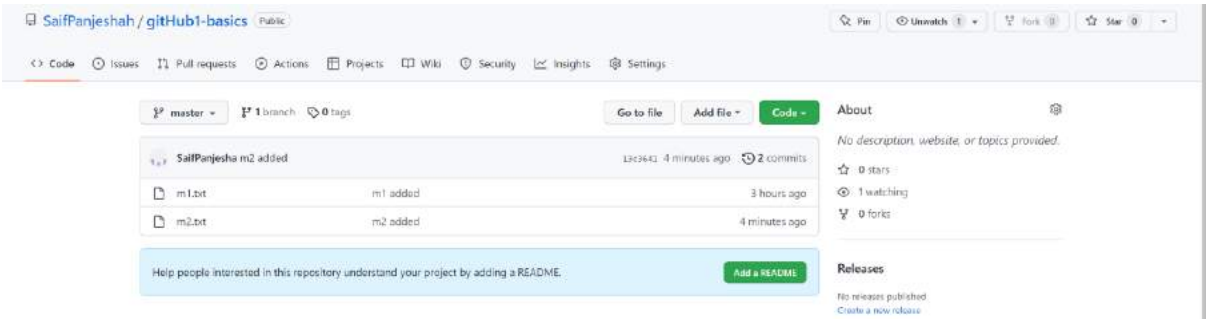


Fig. Successful on GitHub

From Local to remote Understanding the Workflow

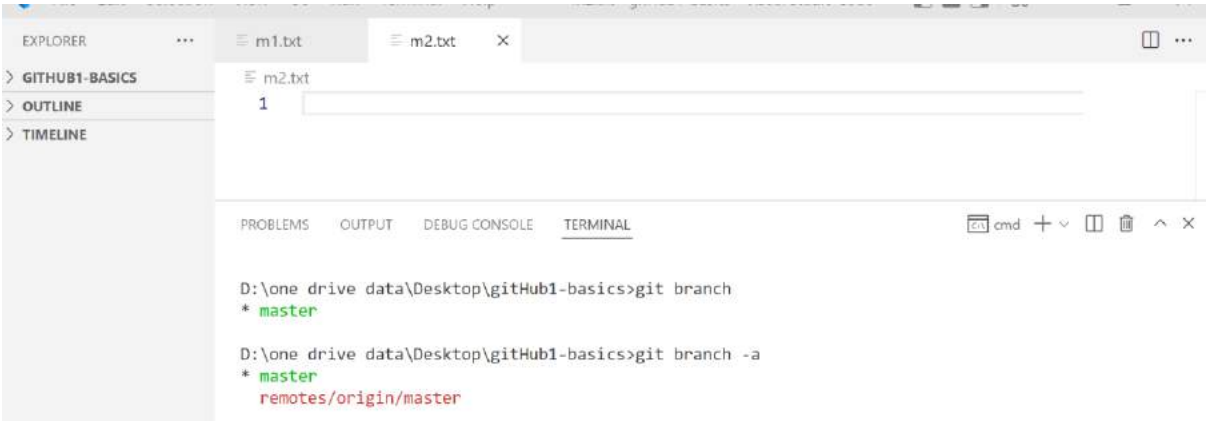


Fig. added remote tracking branch

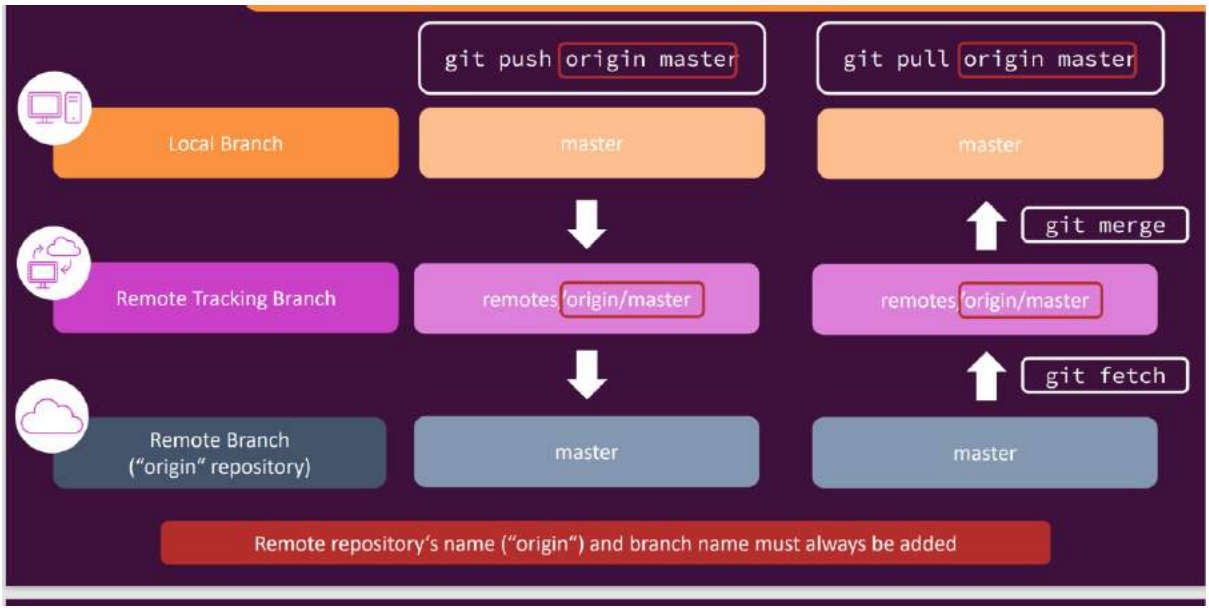


Fig. Local to remote Workflow

Remote Tracking Branches in Practices:

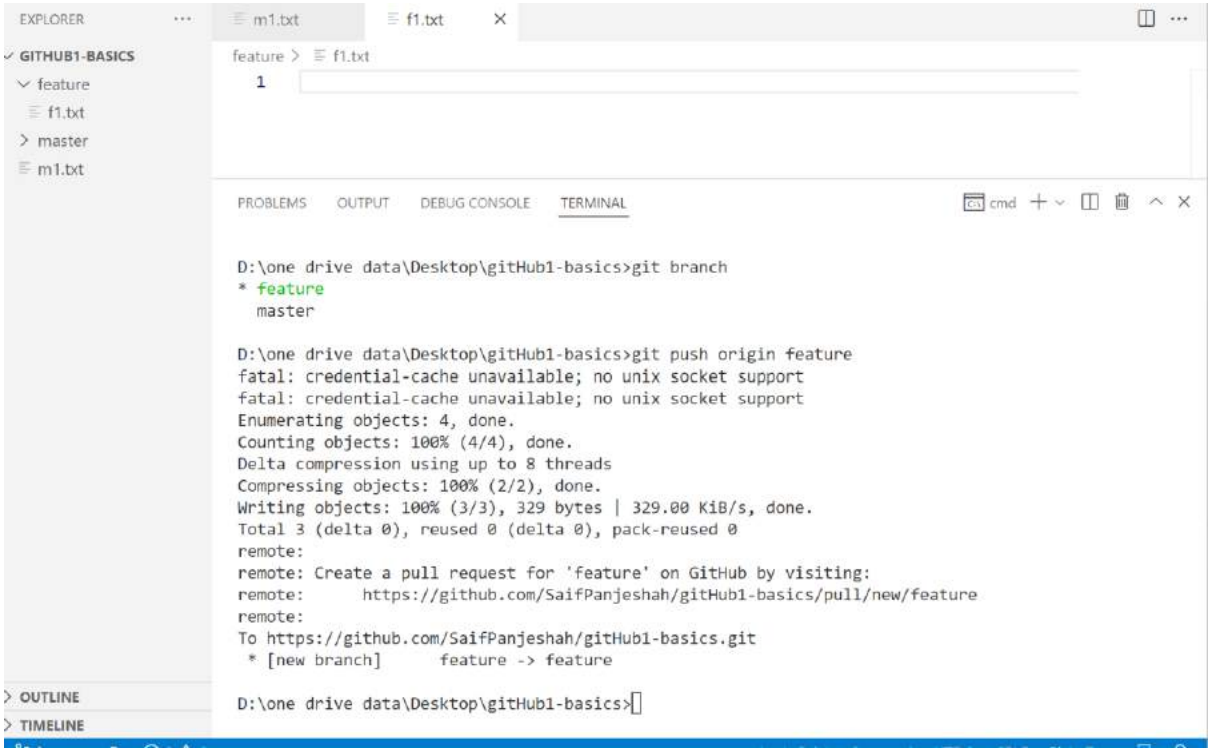


Fig. Created New Feature Branch

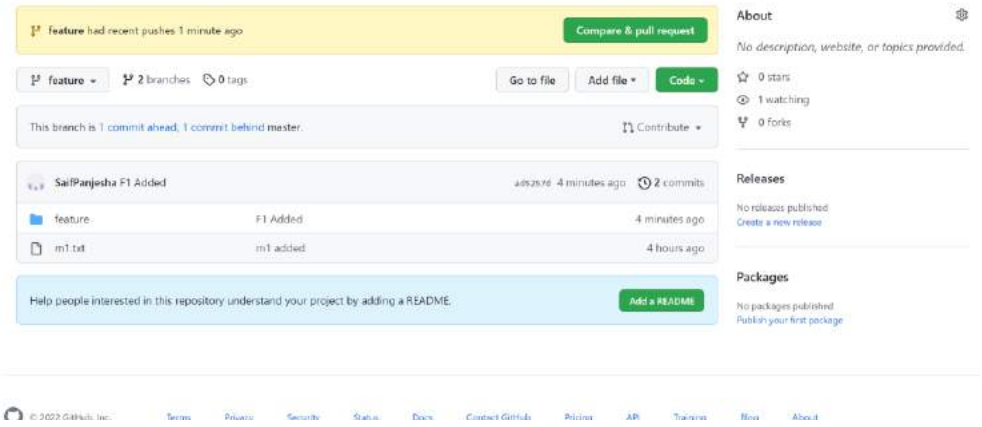


Fig. Successful added on GitHub

D:\one drive data\Desktop\gitHub1-basics>git branch -r

origin/feature

origin/master

D:\one drive data\Desktop\gitHub1-basics>git branch

*** feature**

master

D:\one drive data\Desktop\gitHub1-basics>git branch -a

*** feature**

master

remotes/origin/feature

remotes/origin/master

Creating New Branch on GitHub

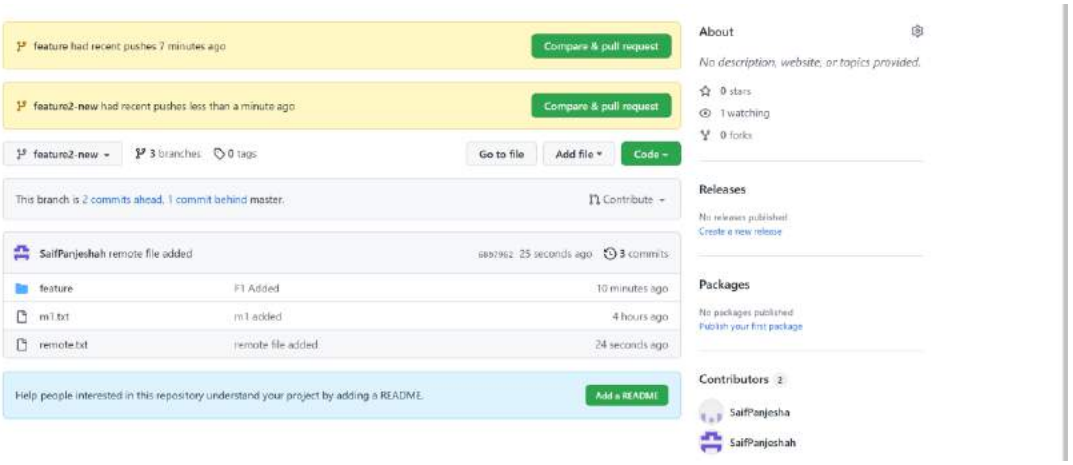


Fig. Created new Branch on Git Hub

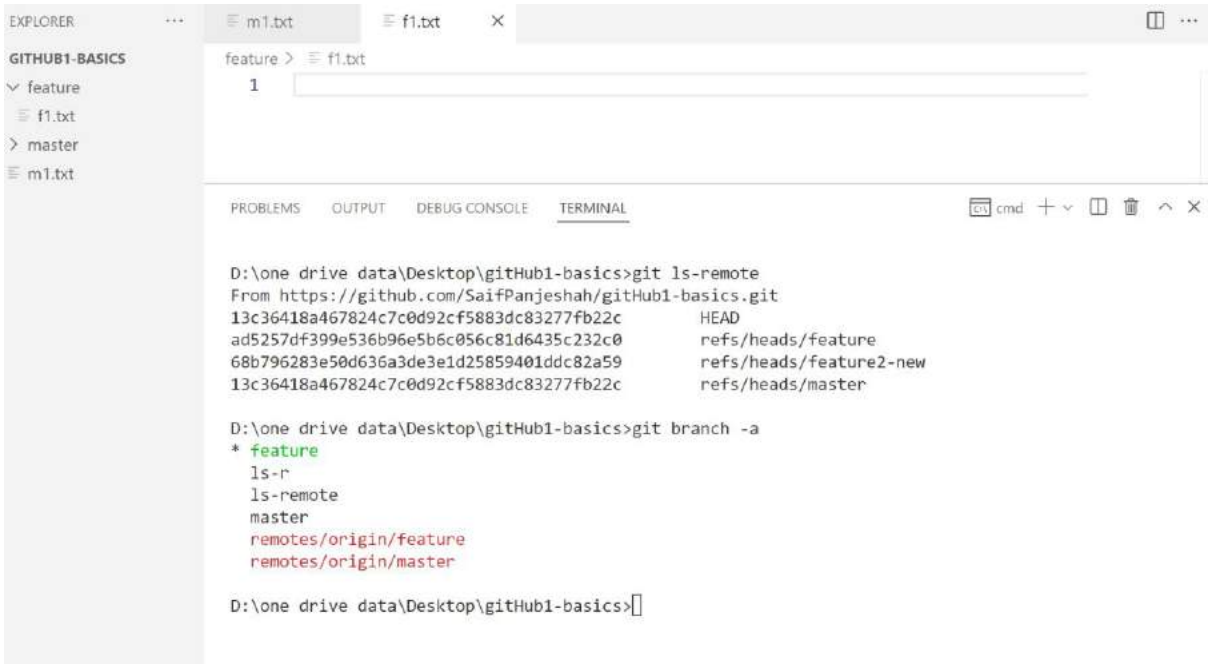


Fig. Shows New Head Branch added in remote repository

How Can View this remote Branch feature2-new in git (local repository)?

Git fetch retrieves the latest state.
Git fetch origin works because it updates the remote tracking branch.

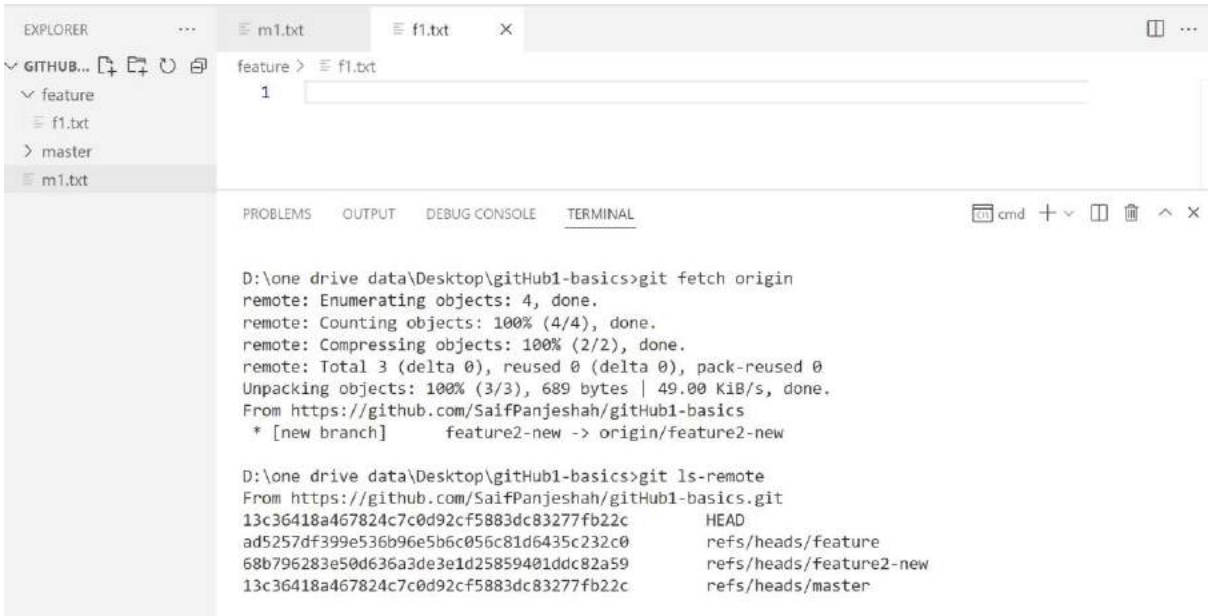


Fig. Shows git fetch remote retrieves latest state

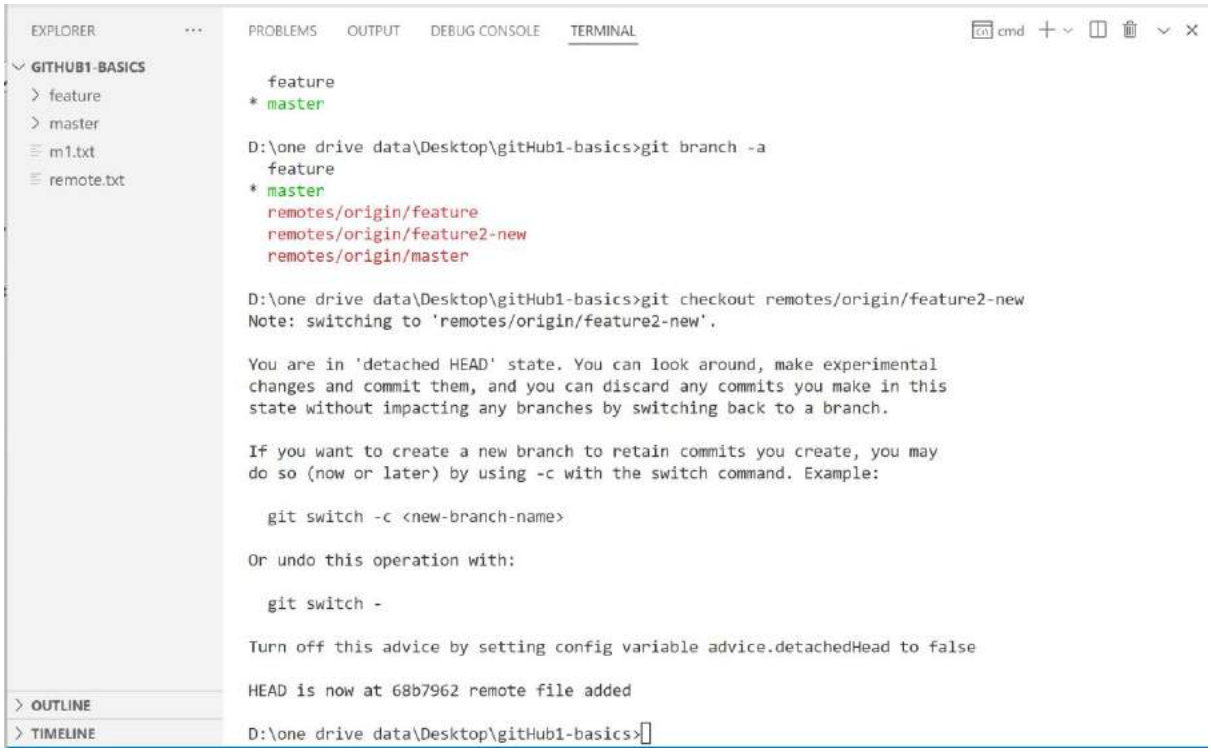


Fig. Shows remote branch only retrieve latest change and in detached mode

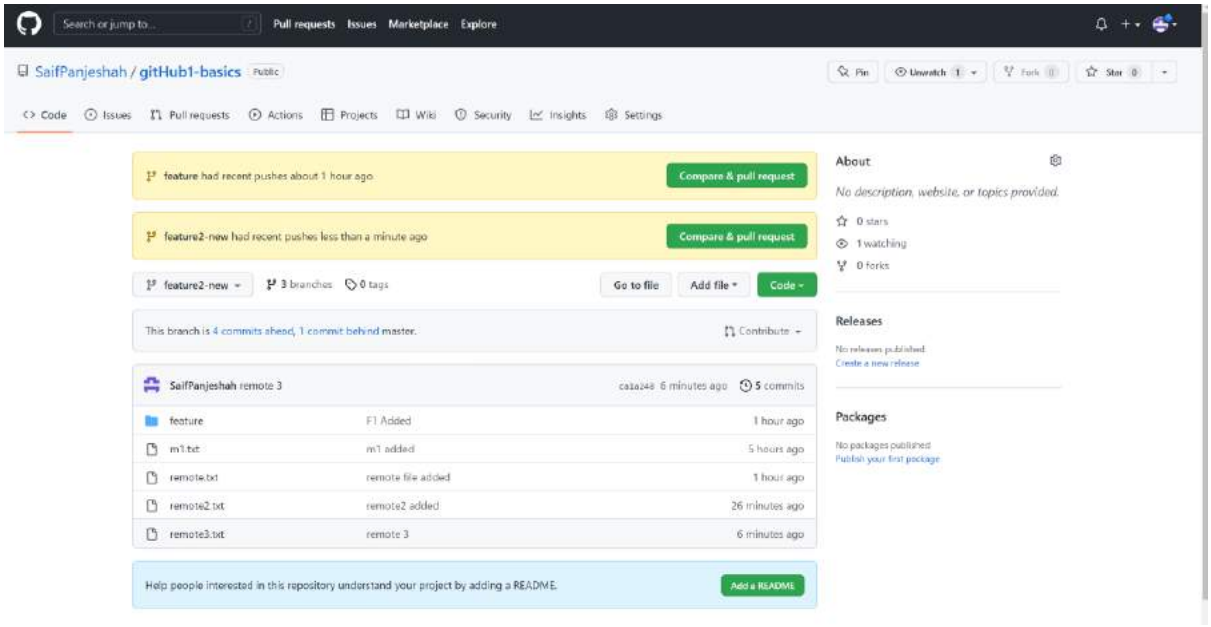


Fig. Shows Feature2 branch in GitHub

Now, to view complete changes we have to use git pull command

git pull → used to fetch and download content from a remote repository and immediately update the local repository to match that content(merge + fetch)

Fig. Shows git pull successful added to local repository

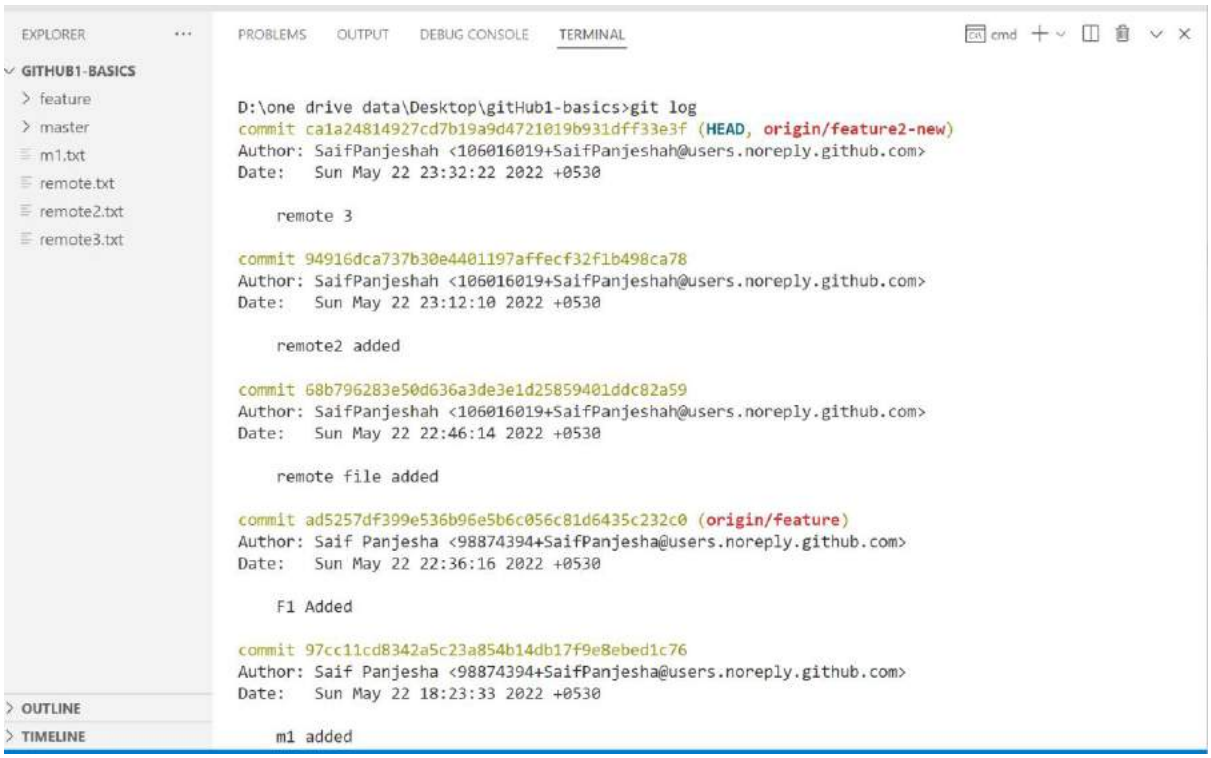


Fig. Shows full commit history of GitHub feature2 -new Branch in local repository

Understanding Local Tracking Branches:

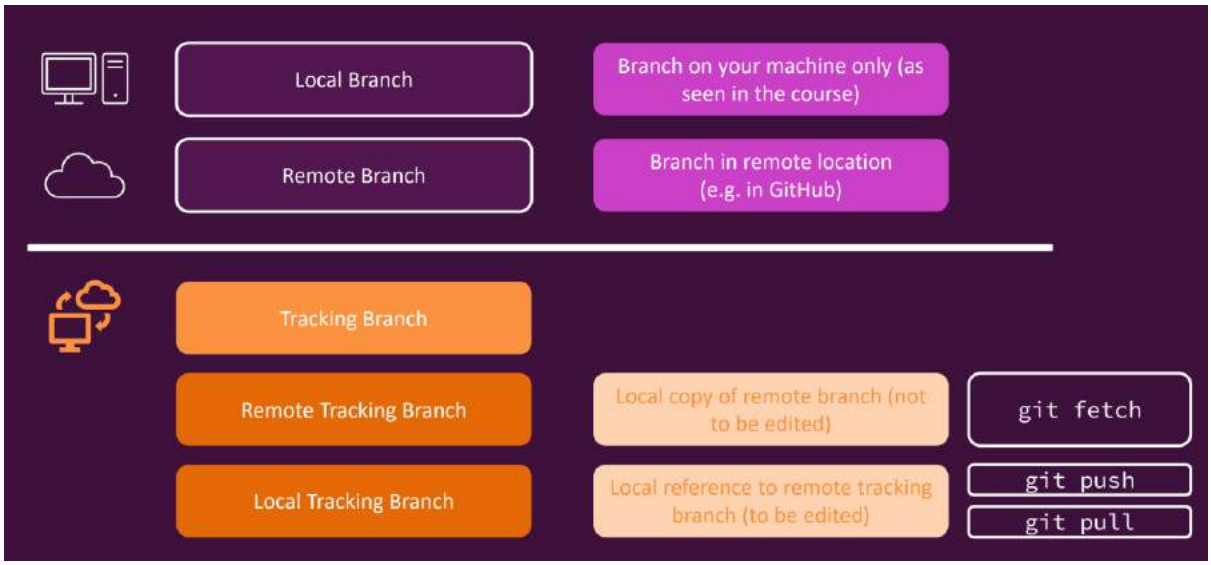


Fig. Overview of Branches

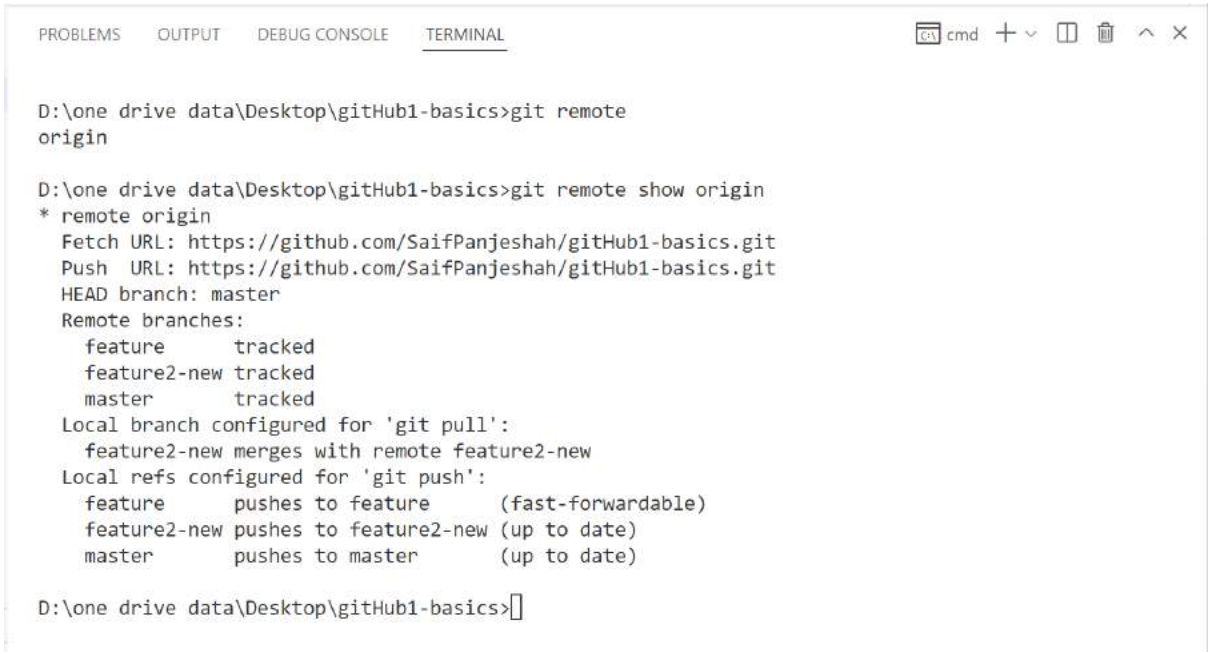


Fig. Commands on terminal

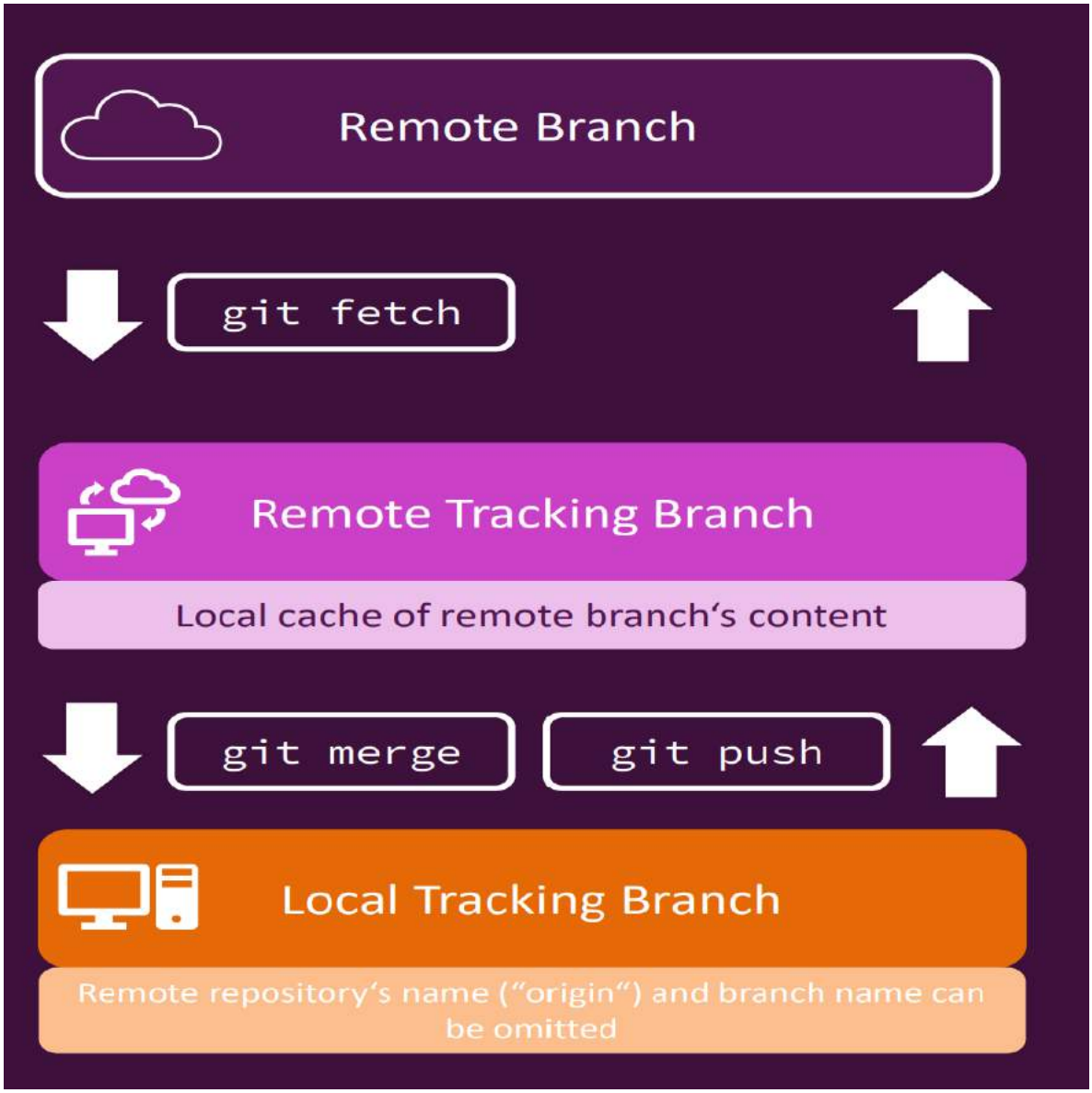


Fig. Local & remote tracking branches

Creating Local Tracking Branches:

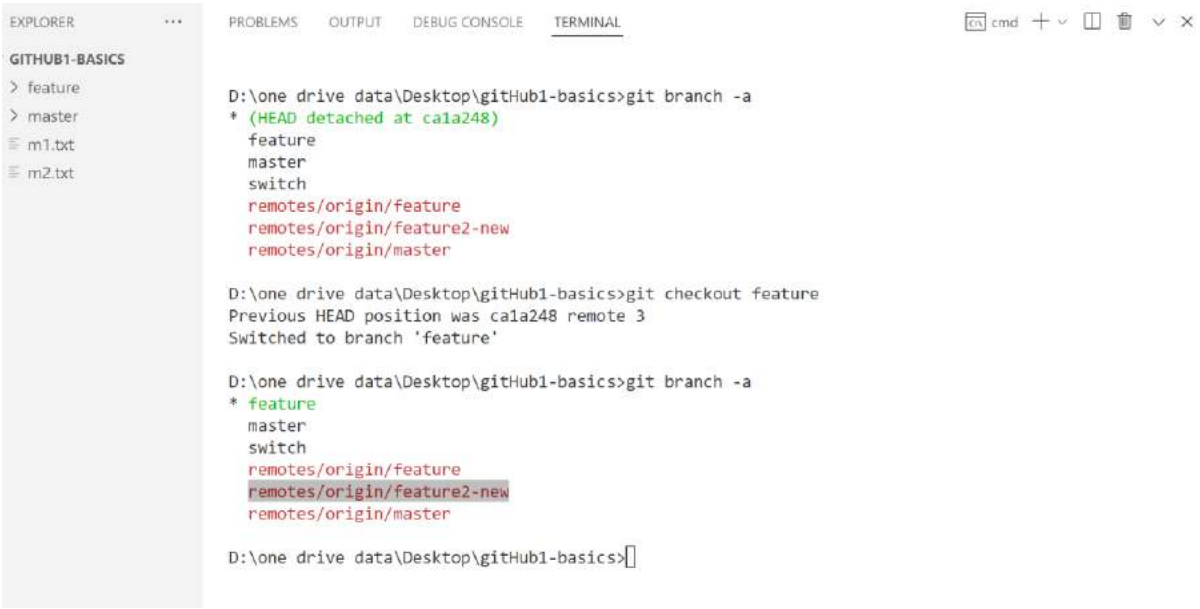


Fig. checking out all branches

Now the goal, is to create local tracking branch of (remotes/origin/feature2-new) remote tracking branch:

git branch --track feature-remote-local origin/feature2-new

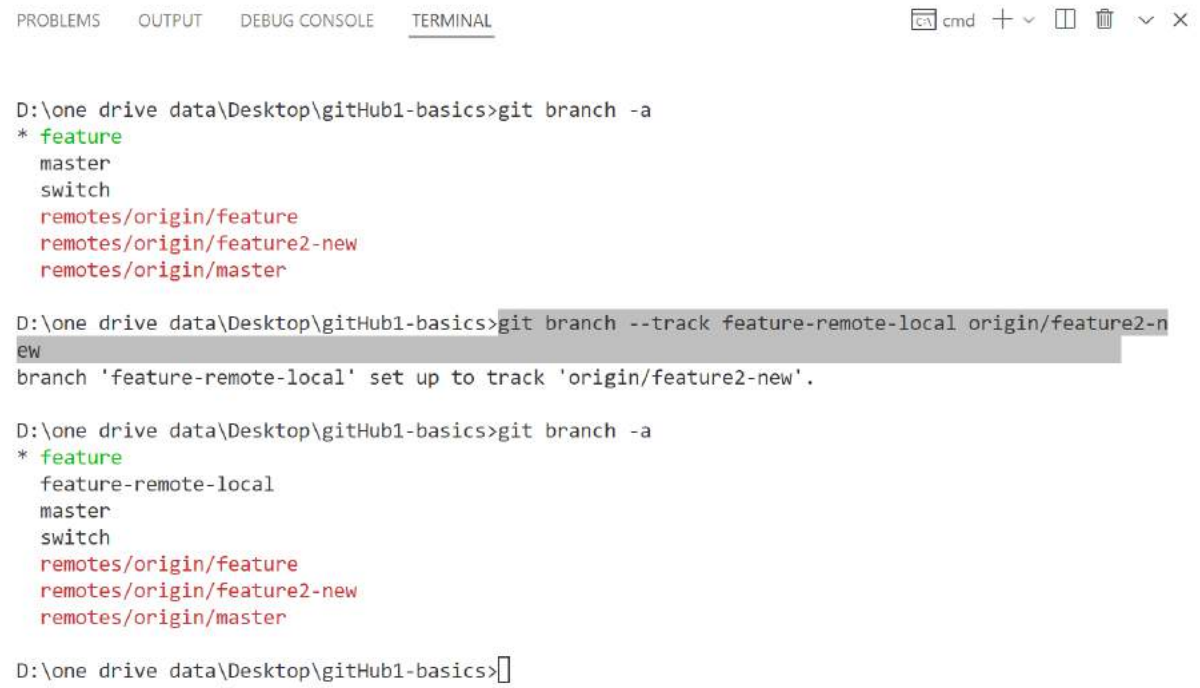


Fig. Local tracking Branches created as structured

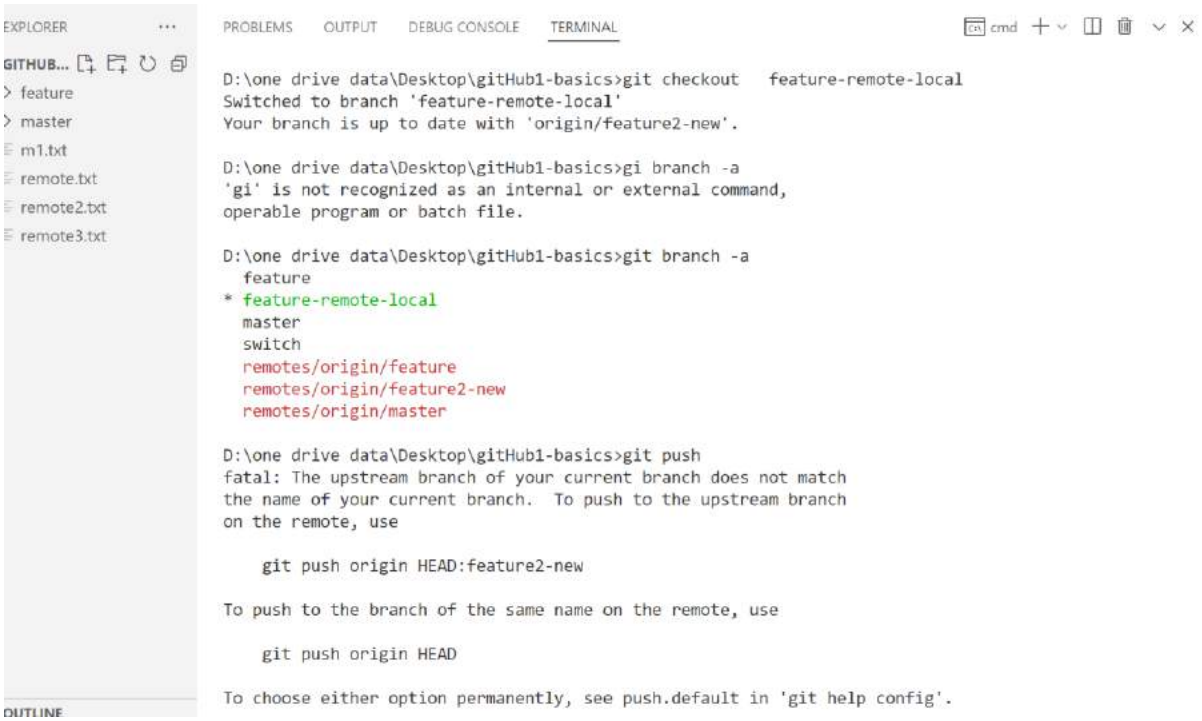


Fig. after pushing the upstream branches not matching

Now, to avoid this fatal error we have to create same branch as named in remote tracking branches

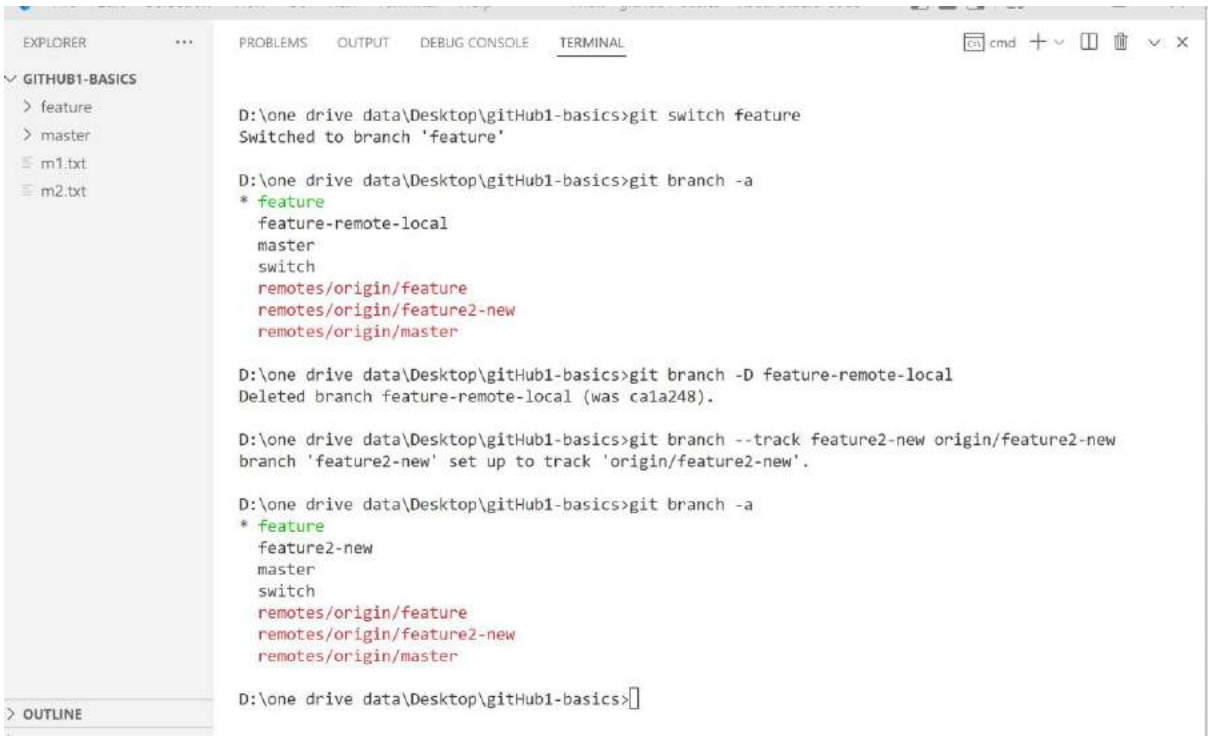


Fig. created same name as remote tracking branch

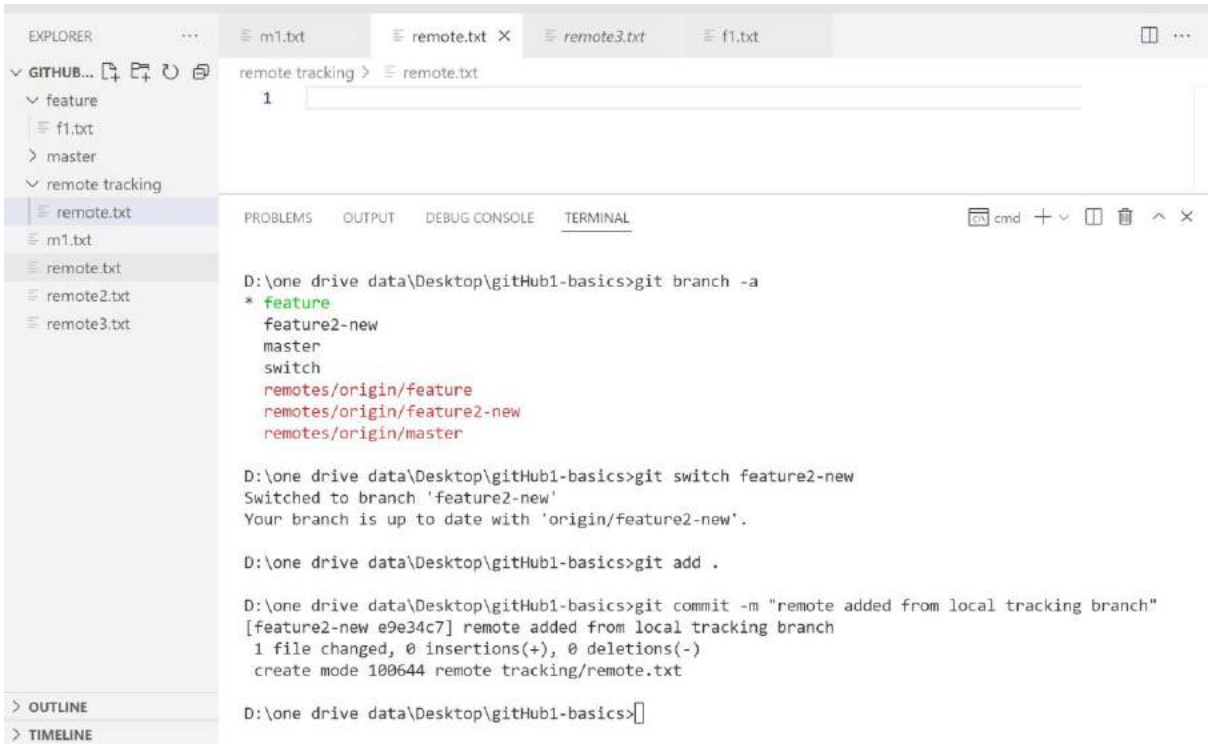


Fig. Added new file remote.txt

Now push, on remote tracking branches:

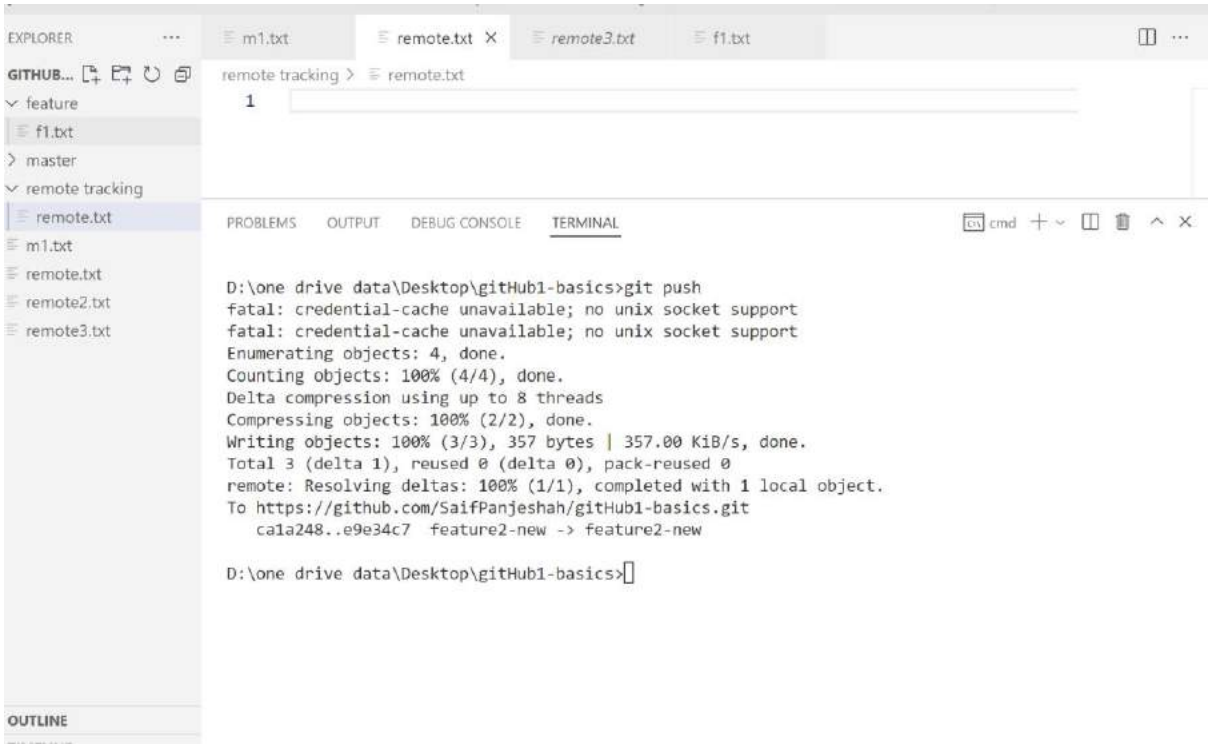


Fig. Success on remote tracking branches

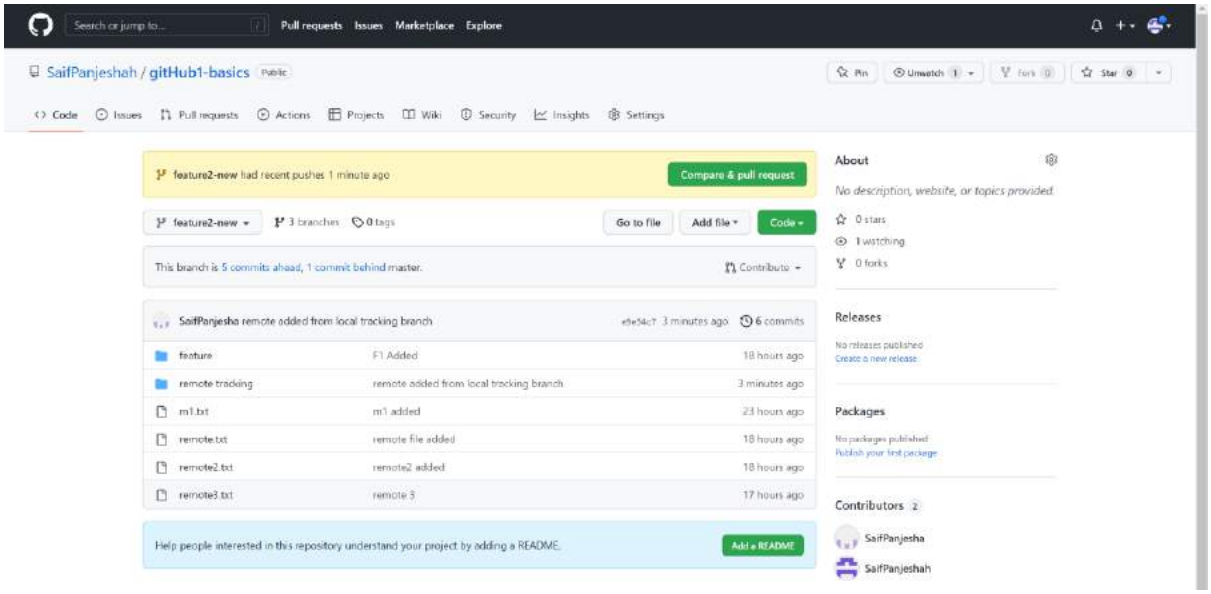


Fig. Successful on remote added from local tracking branches

Now let's try for pull from remote tracking branches to local tracking branches

Created New File :

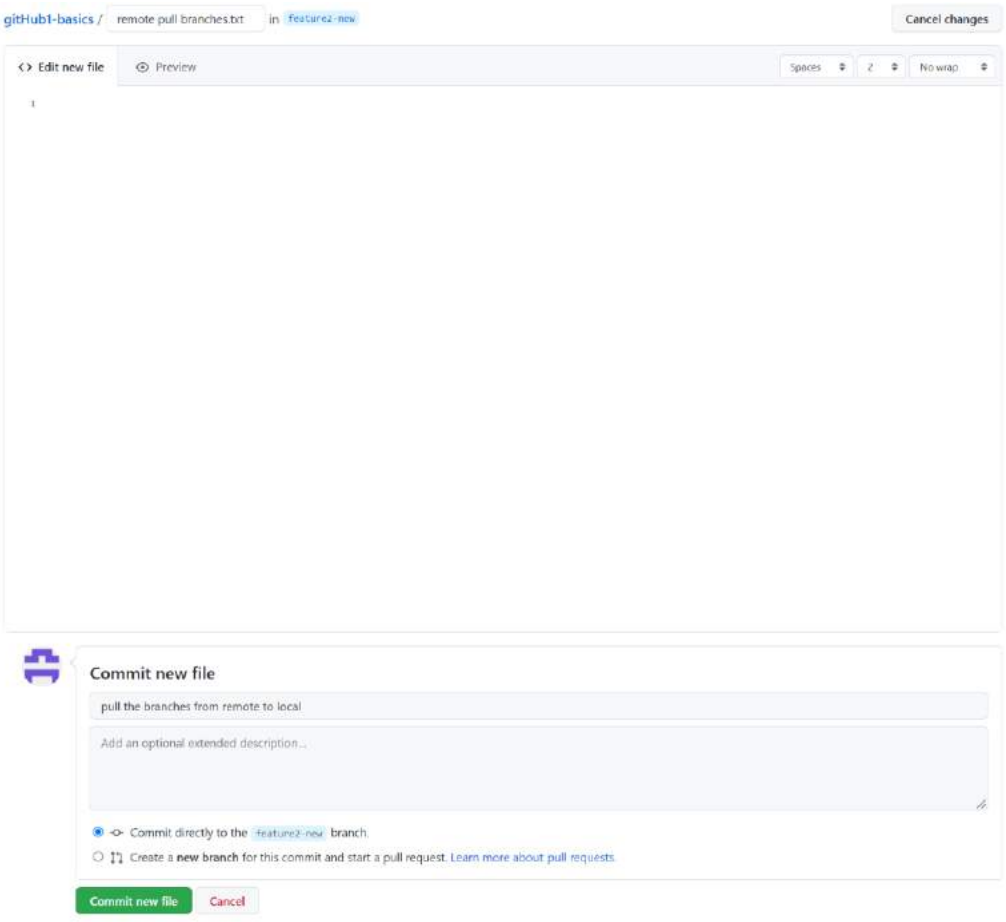


Fig. Created a new File on remote tracking branches

git branch -vv: List local tracking branches and their remotes

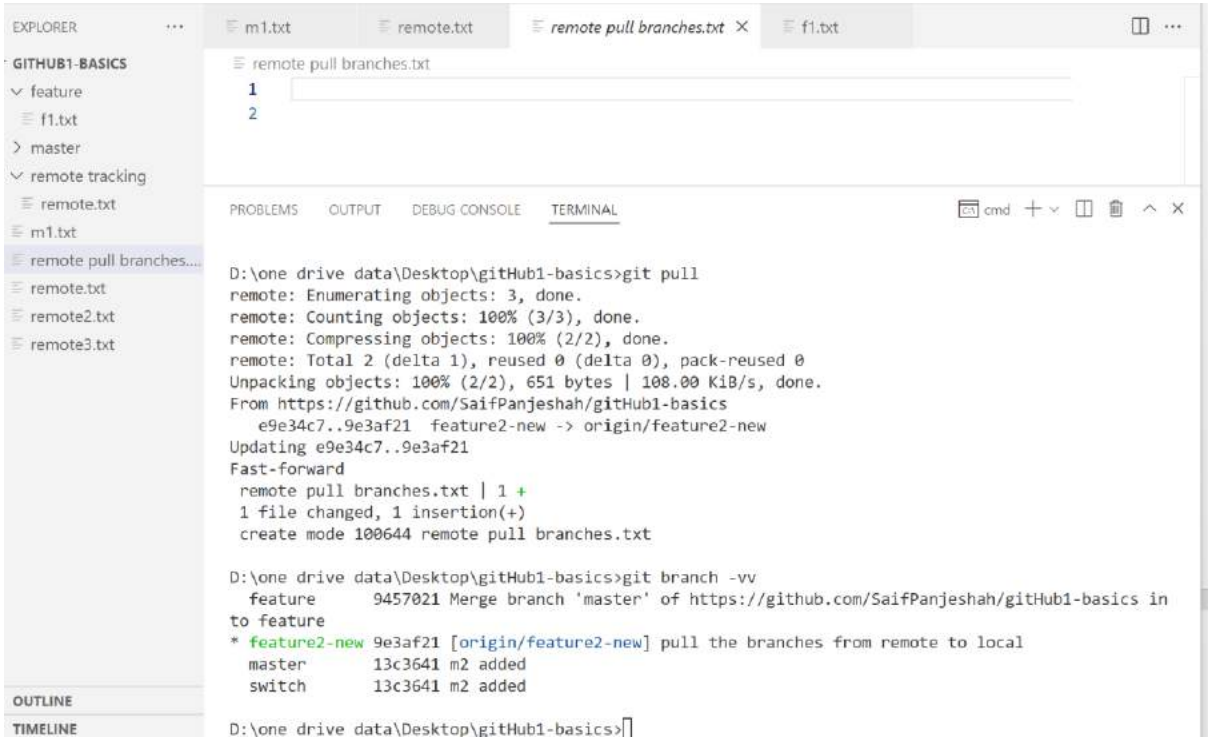


Fig. Successful added to local tracking branches from remote vice versa

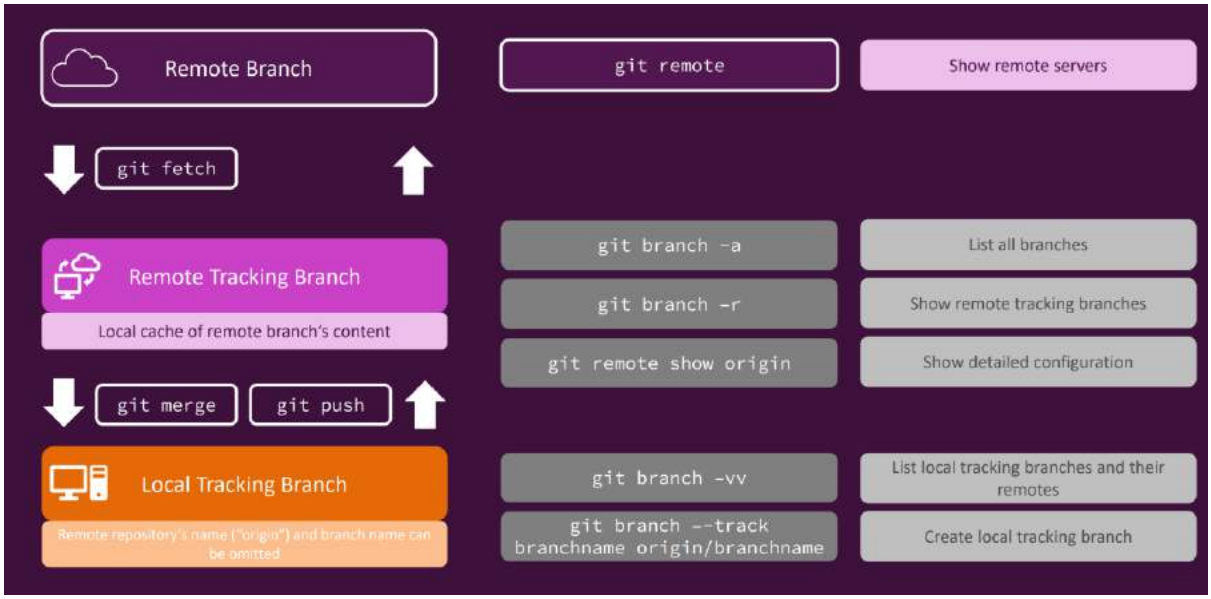


Fig. Local & Remote Tracking Branches Commands

Cloning a remote repository:

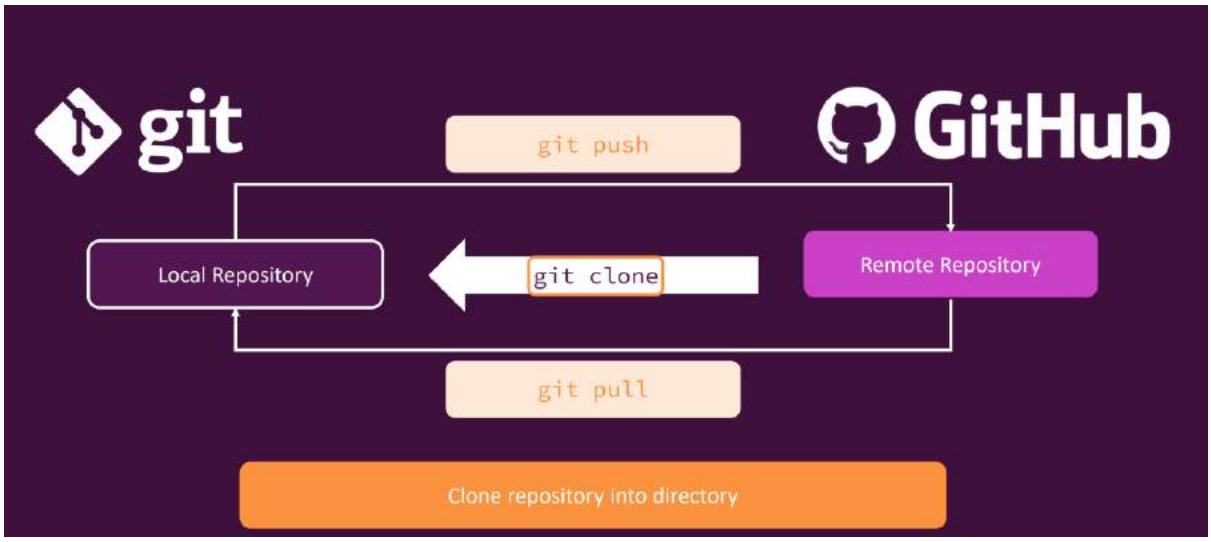


Fig. Shows How to Clone a remote repository

git clone is a git command line utility used to target an existing repository and create a clone, or copy of the target repository.

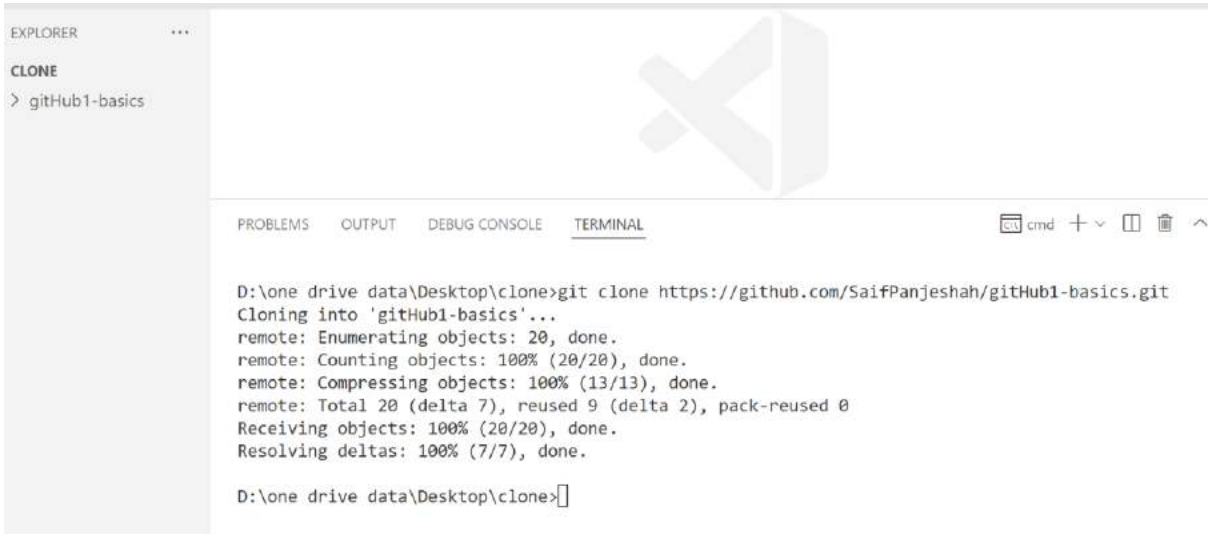


Fig. Cloning repository into folder

```
D:\one drive data\Desktop\clone>git status
fatal: not a git repository (or any of the parent directories): .git
```

Fig. Fatal error while using git commands

How to resolve this fatal error?

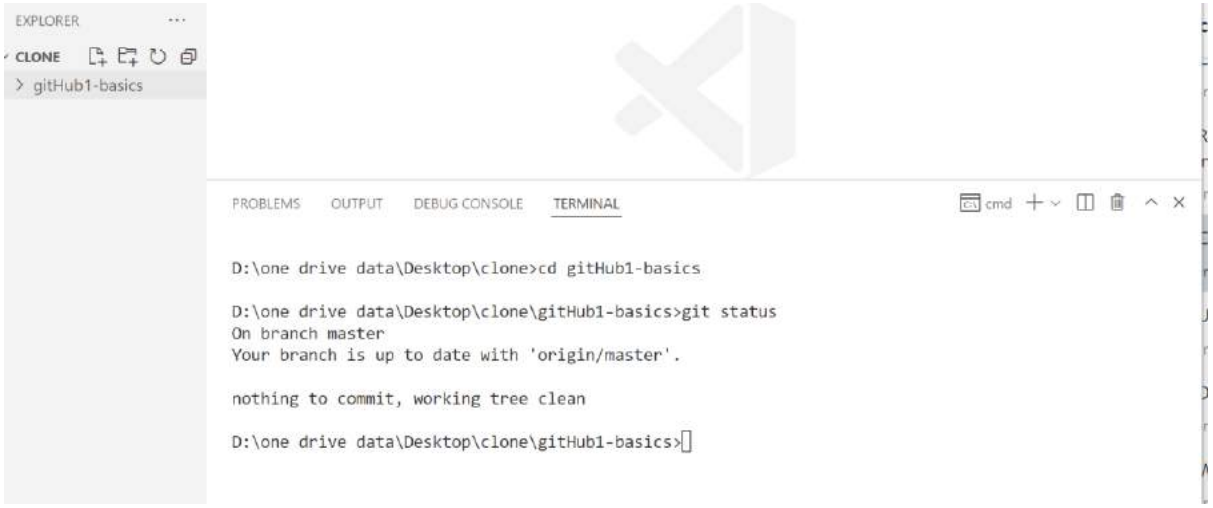


Fig. resolve fatal error by changing directory to repository

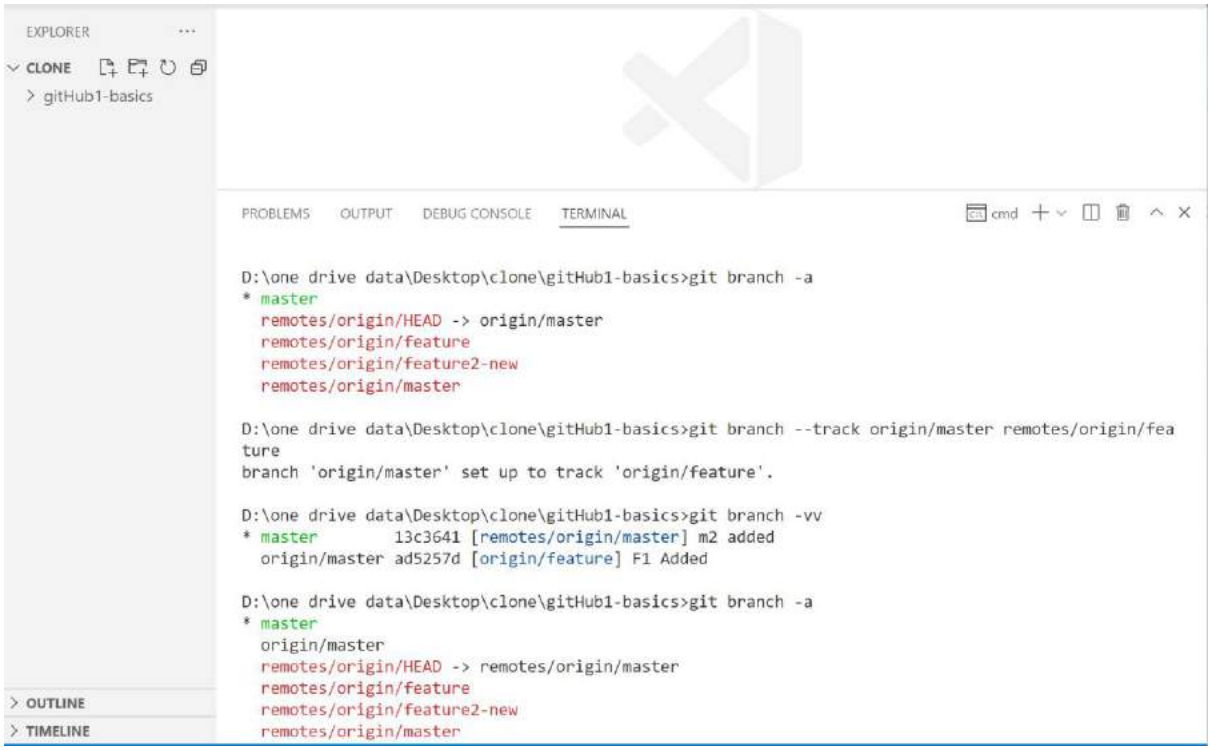


Fig. Changing head to feature branch

Let’s Create a New Branch in our local branch tracking:

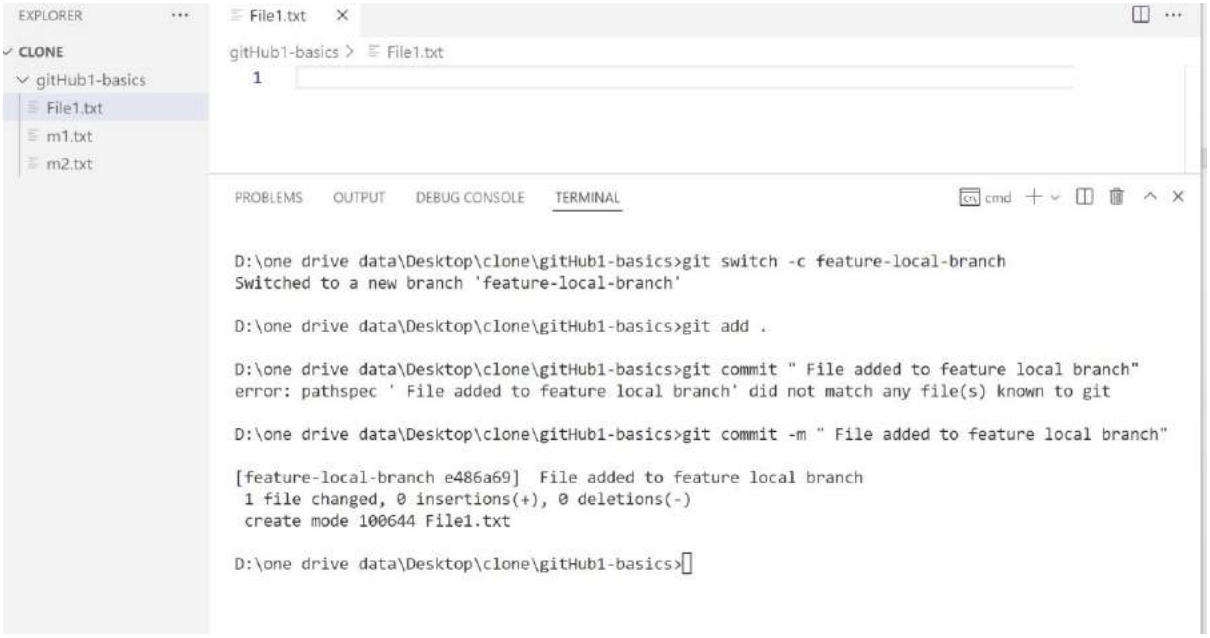


Fig. Created feature-local-branch

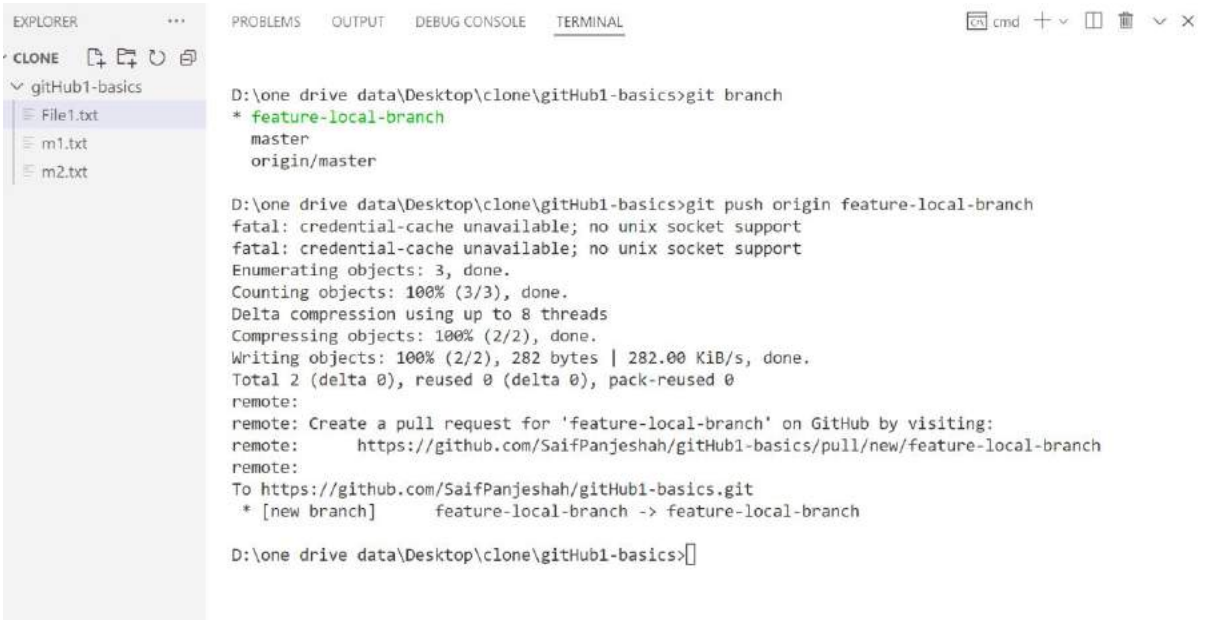


Fig. Successful push on remote branch

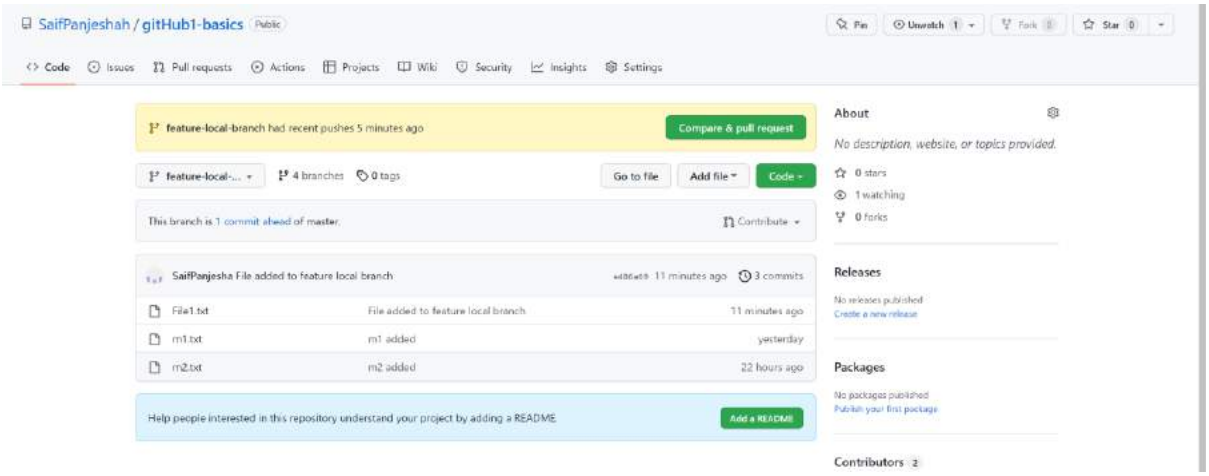


Fig. Successful added on remote branch GitHub

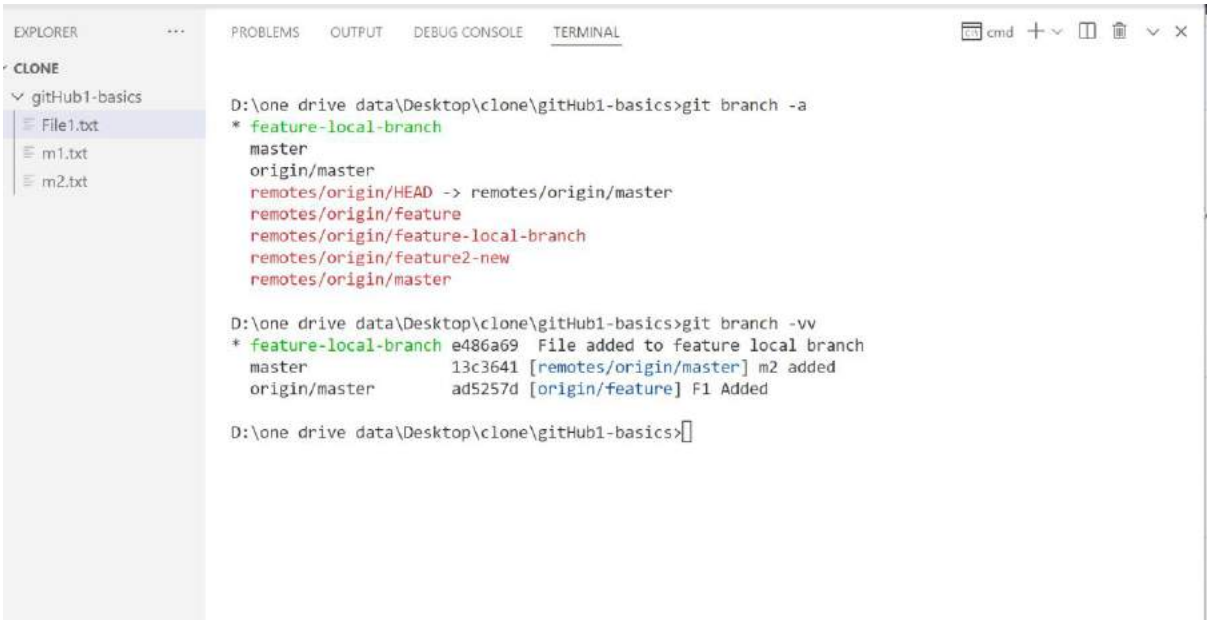


Fig. Shows all tracking branches

Now, our feature-local-branch refer to any kind of well remote tracking branch? So How Can we turn into local tracking branch?

D:\one drive data\Desktop\clone\gitHub1-basics>git branch -vv
*** feature-local-branch e486a69 File added to feature local branch**
master 13c3641 [remotes/origin/master] m2 added
origin/master ad5257d [origin/feature] F1 Added

D:\one drive data\Desktop\clone\gitHub1-basics> git switch master
Switched to branch 'master'
Your branch is up to date with 'remotes/origin/master'.

D:\one drive data\Desktop\clone\gitHub1-basics>git branch -D feature-local-branch
Deleted branch feature-local-branch (was e486a69).

D:\one drive data\Desktop\clone\gitHub1-basics>

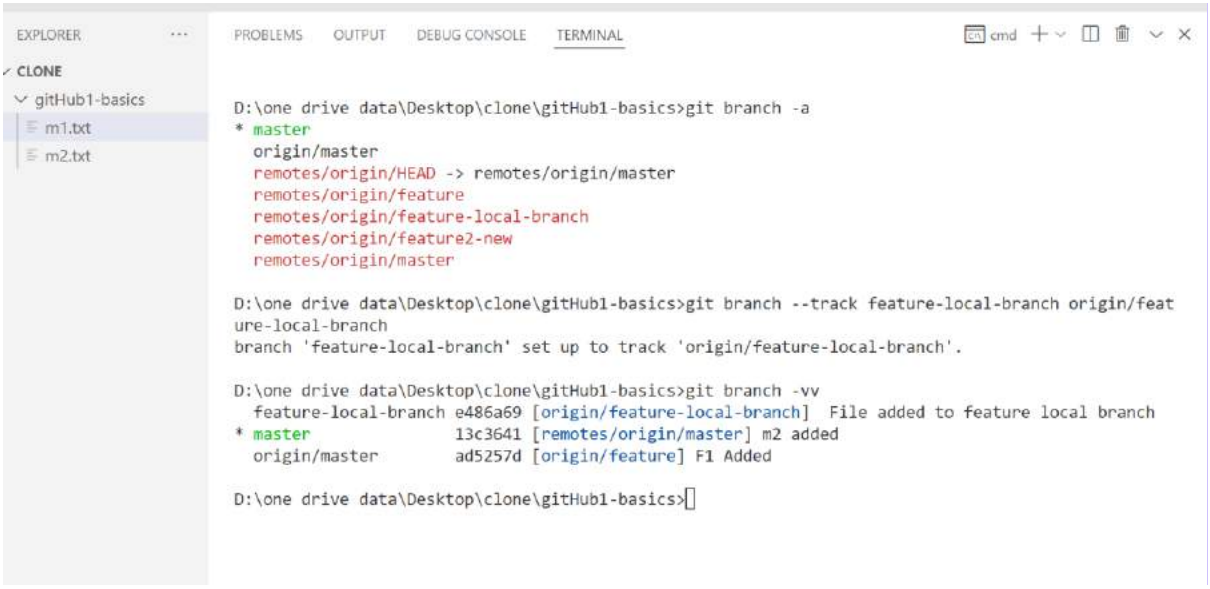


Fig. Created Local tracking branch

```
D:\one drive data\Desktop\clone\gitHub1-basics>git branch -a
  feature-local-branch
* master
  origin/master
  remotes/origin/HEAD -> remotes/origin/master
  remotes/origin/feature
  remotes/origin/feature-local-branch
  remotes/origin/feature2-new
  remotes/origin/master

D:\one drive data\Desktop\clone\gitHub1-basics>git switch   feature-local-branch
Switched to branch 'feature-local-branch'
Your branch is up to date with 'origin/feature-local-branch'.

D:\one drive data\Desktop\clone\gitHub1-basics>
```

Fig. Success Local tracking branch

Understanding the Upstream:

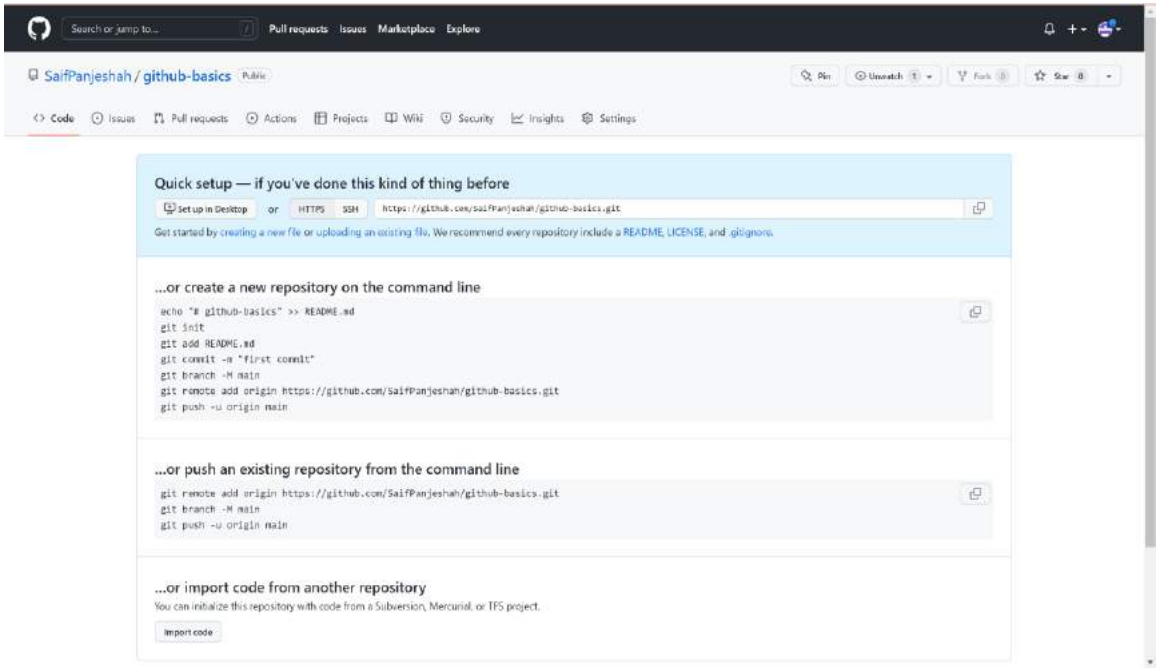


Fig. Checkout for "-u" Upstream

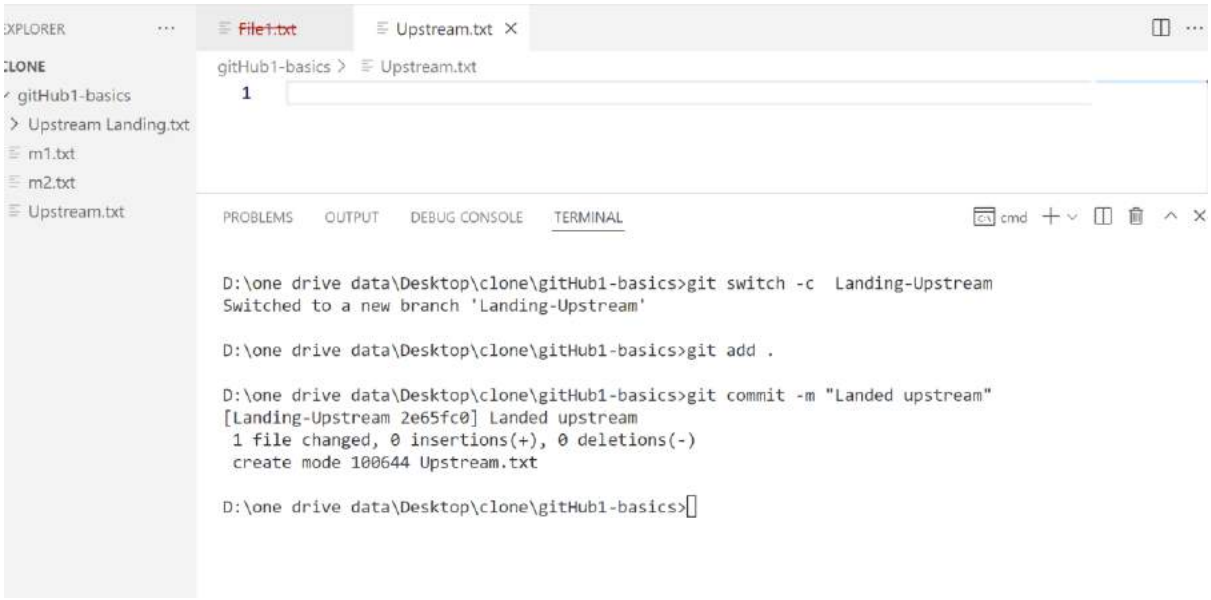


Fig. Created upstream branch

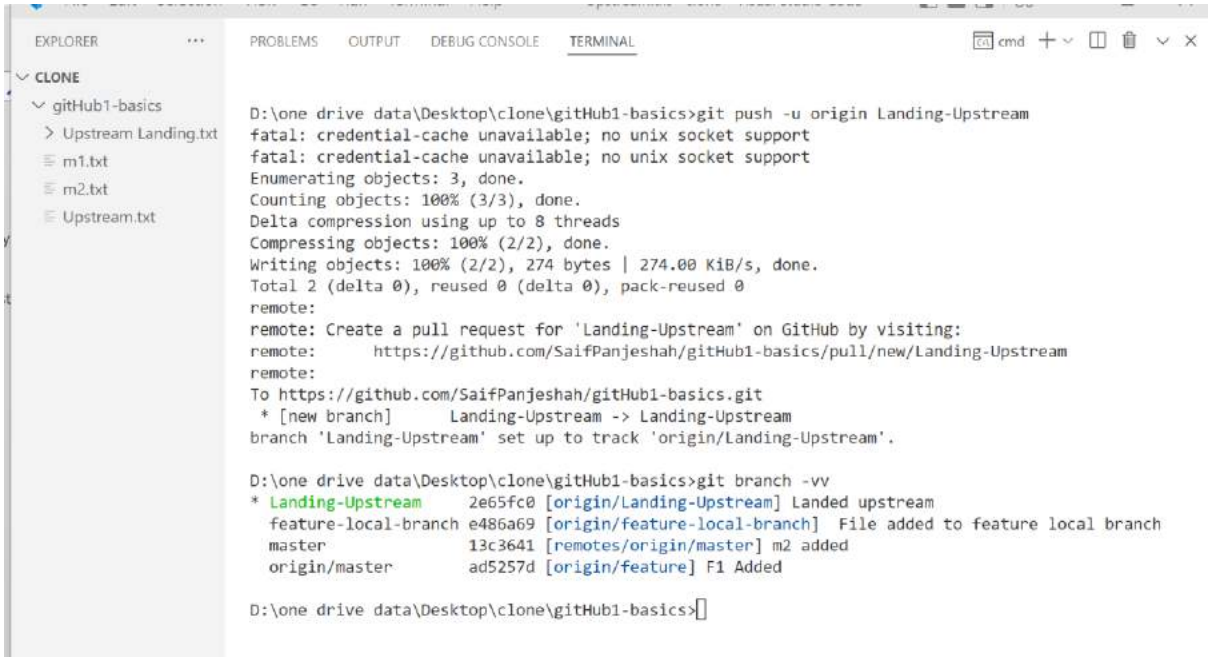


Fig. Successful added on remote without tracking the branches

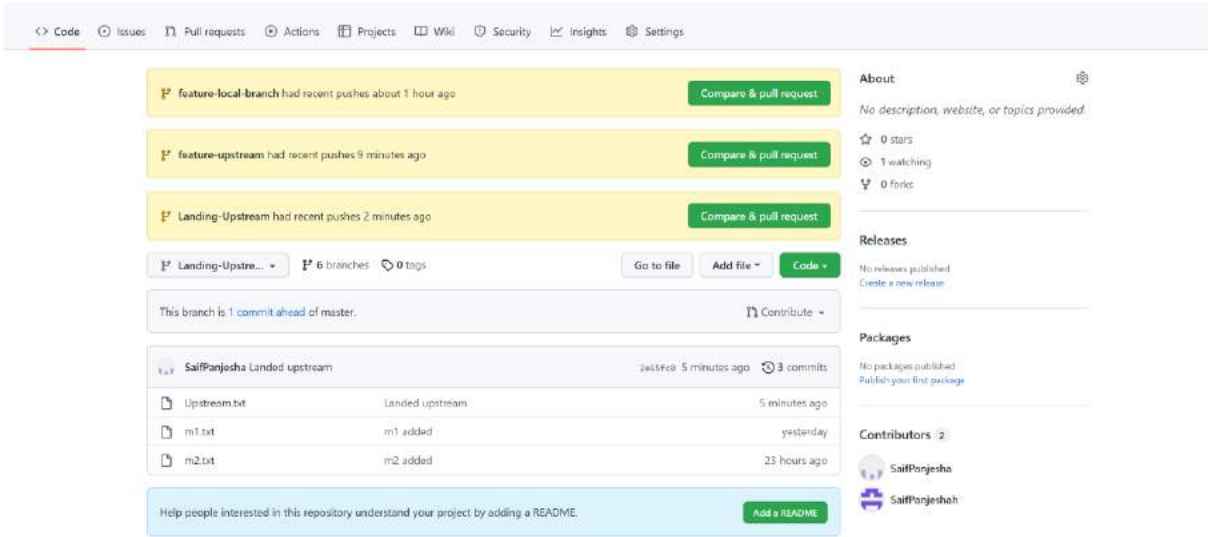


Fig. Successful Upstream.txt on GitHub

Deleting Remote Branches & Public Commits:

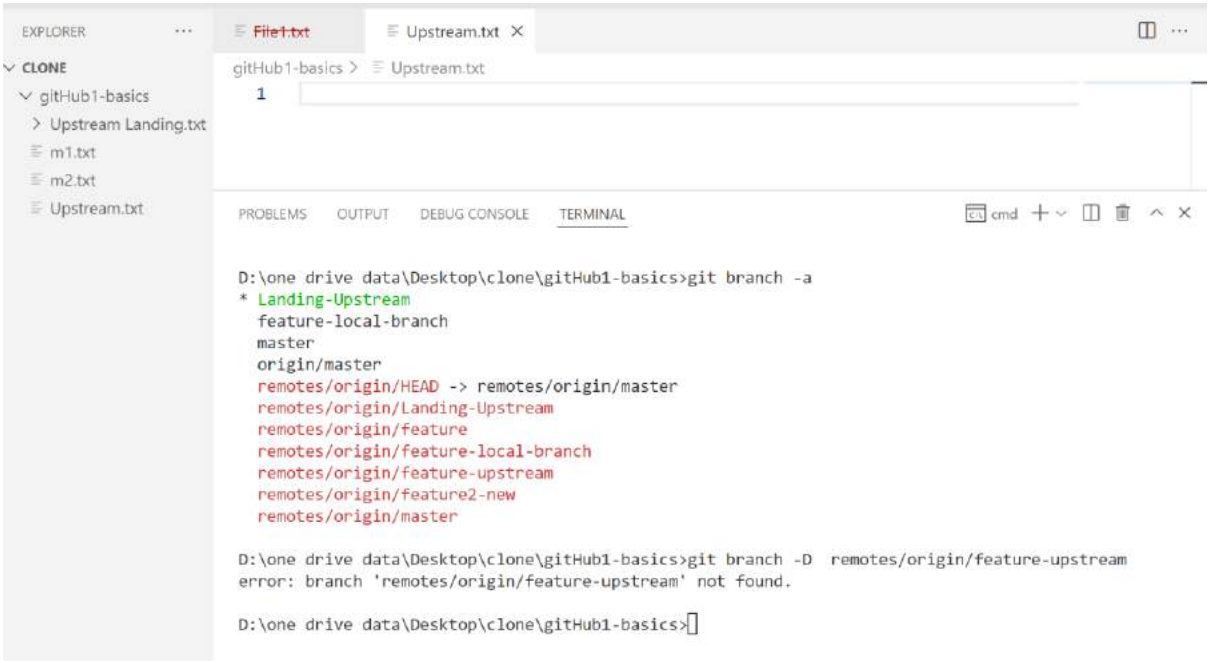


Fig. Not Working deletion

How can we delete now?

git branch - -delete - - remote origin/feature-upstream

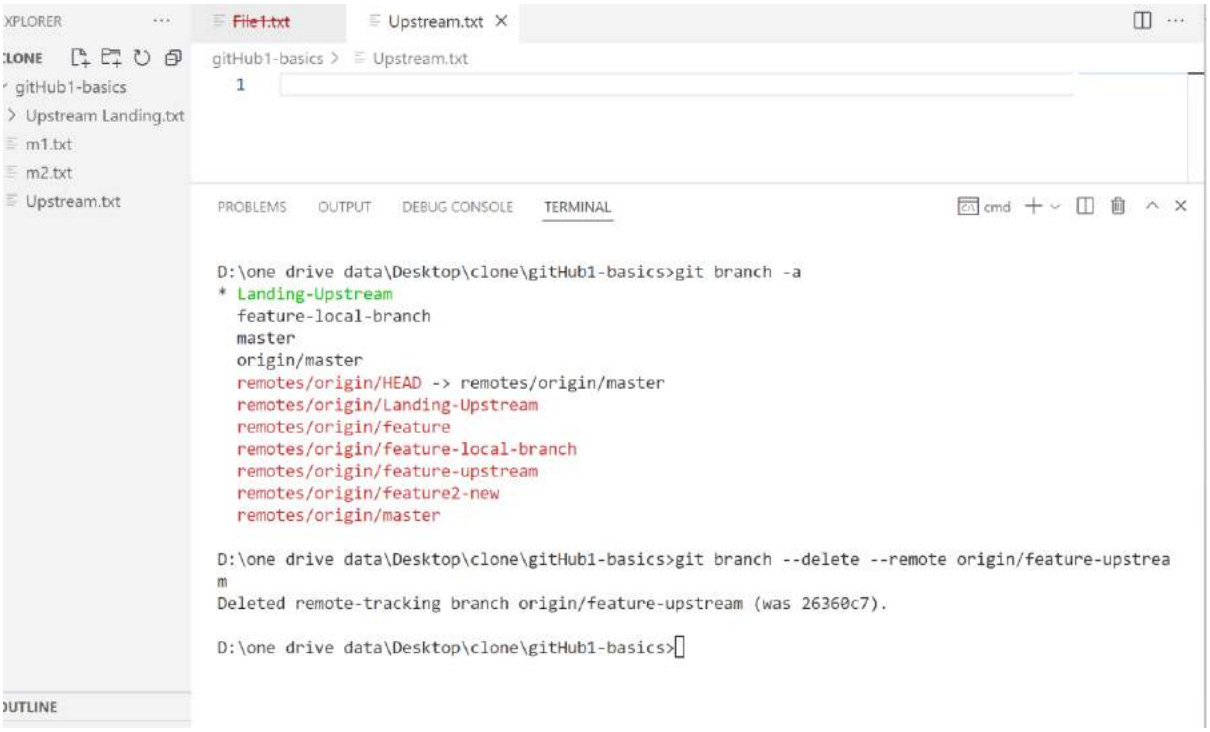


Fig. Deleted branches

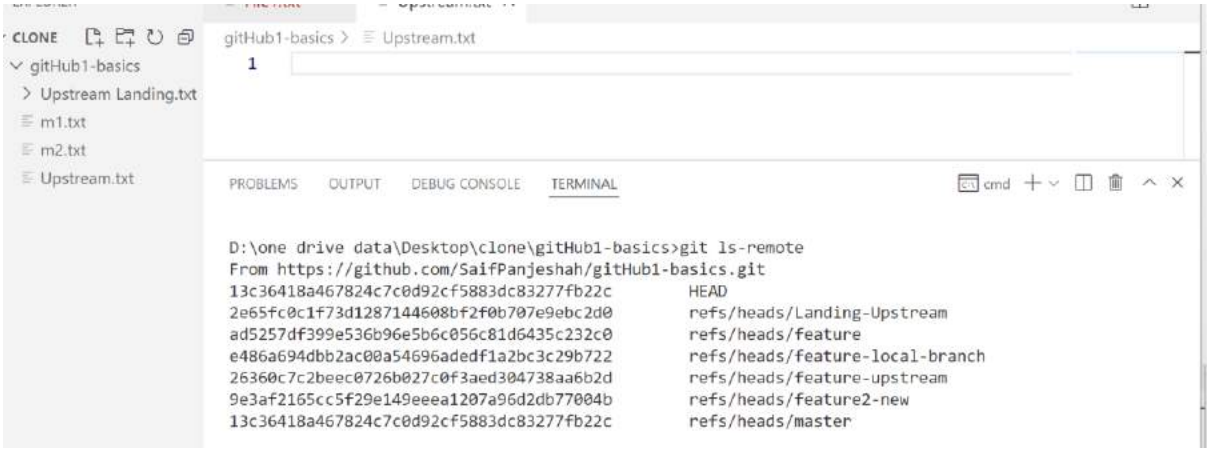


Fig. ls-remotes shows that feature-upstream is in remote

How can we delete now?

```
D:\one drive data\Desktop\clone\gitHub1-basics>git push origin --delete feature-upstream
fatal: credential-cache unavailable; no unix socket support
fatal: credential-cache unavailable; no unix socket support
To https://github.com/SaifPanjeshah/gitHub1-basics.git
 - [deleted]         feature-upstream

D:\one drive data\Desktop\clone\gitHub1-basics>git ls-remote
From https://github.com/SaifPanjeshah/gitHub1-basics.git
13c36418a467824c7c0d92cf5883dc83277fb22c        HEAD
2e65fc0c1f73d1287144608bf2f0b707e9ebc2d0        refs/heads/Landing-Upstream
ad5257df399e536b96e5b6c056c81d6435c232c0        refs/heads/feature
e486a694dbb2ac00a54696adedf1a2bc3c29b722        refs/heads/feature-local-branch
9e3af2165cc5f29e149eeea1207a96d2db77004b        refs/heads/feature2-new
13c36418a467824c7c0d92cf5883dc83277fb22c        refs/heads/master
```

Fig. Successful deleted

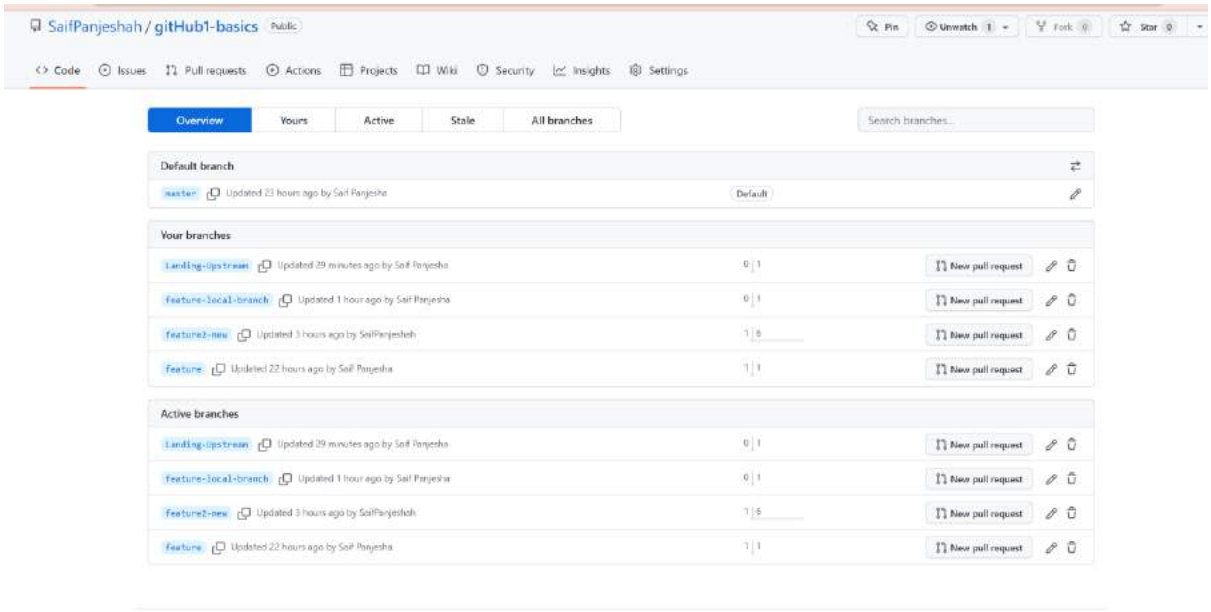


Fig. feature-upstream successful deleted on GitHub

How to delete Public Commits:

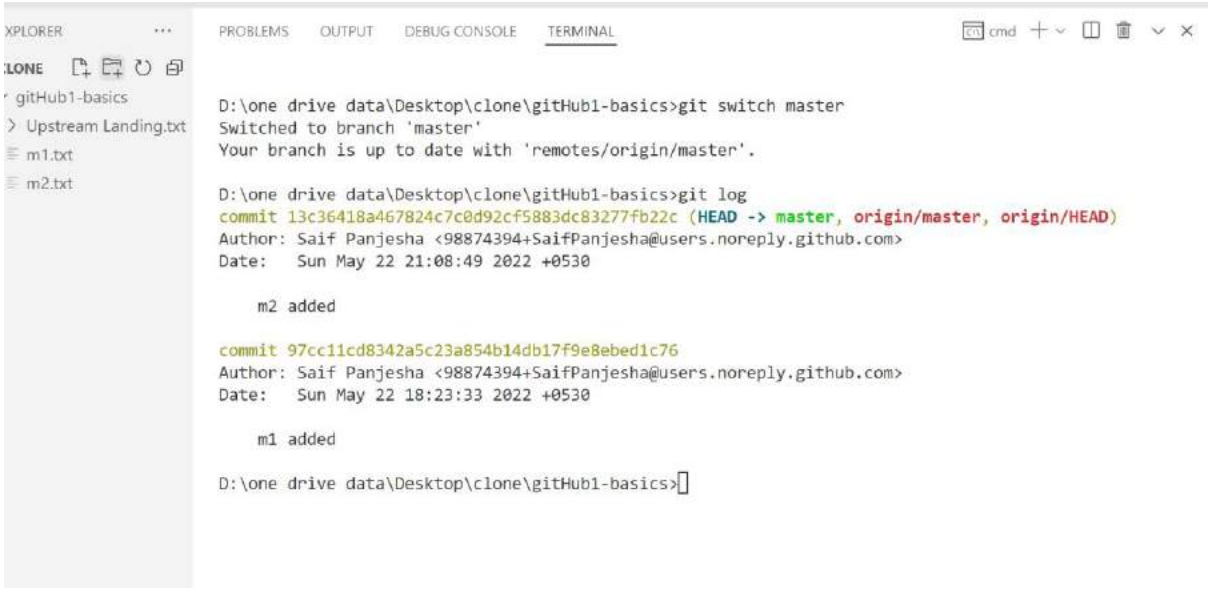


Fig. Public Commits

```
D:\one drive data\Desktop\clone\gitHub1-basics>git log
commit 13c36418a467824c7c0d92cf5883dc83277fb22c (HEAD -> master, origin/master, origin/HEAD)
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Sun May 22 21:08:49 2022 +0530

    m2 added

commit 97cc11cd8342a5c23a854b14db17f9e8ebed1c76
Author: Saif Panjesha <98874394+SaifPanjesha@users.noreply.github.com>
Date:   Sun May 22 18:23:33 2022 +0530

    m1 added

D:\one drive data\Desktop\clone\gitHub1-basics>git reset --hard HEAD~1
HEAD is now at 97cc11c m1 added

D:\one drive data\Desktop\clone\gitHub1-basics>git push origin master
fatal: credential-cache unavailable; no unix socket support
fatal: credential-cache unavailable; no unix socket support
To https://github.com/SaifPanjeshah/gitHub1-basics.git
 ! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to 'https://github.com/SaifPanjeshah/gitHub1-basics.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

D:\one drive data\Desktop\clone\gitHub1-basics>
```

Fig. Public Commits deleted unable to push

How can we delete this commit?

```
D:\one drive data\Desktop\clone\gitHub1-basics>git push --force  origin master
fatal: credential-cache unavailable; no unix socket support
fatal: credential-cache unavailable; no unix socket support
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/SaifPanjeshah/gitHub1-basics.git
 + 13c3641...97cc11c master -> master (forced update)

D:\one drive data\Desktop\clone\gitHub1-basics>git pull origin master
From https://github.com/SaifPanjeshah/gitHub1-basics
 * branch            master     -> FETCH_HEAD
Already up to date.

D:\one drive data\Desktop\clone\gitHub1-basics>
```

Fig. Successful push deleted commits and pull master

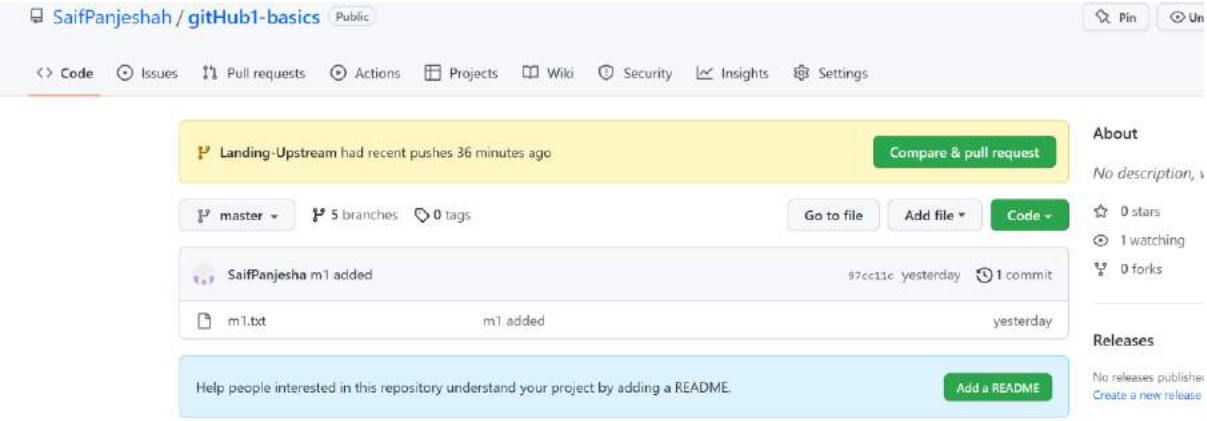


Fig. Success on Deletion GitHub

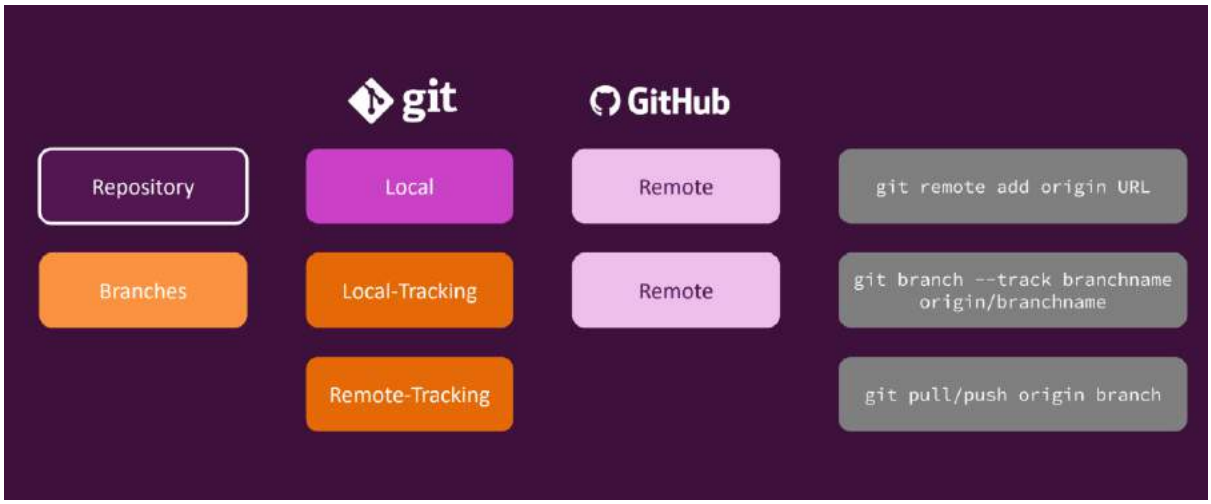


Fig. Wrap up summary

Useful Resources & Links

GitHub official website => **https://github.com/**

GitHub pricing => **https://github.com/pricing**

GitHub Deep Dive – Collaboration & Contribution

Module Introduction:

Fig. Module Introduction

Module Content:

1. **Understand GitHub Accounts & Repository Types**
2. **Collaborating to GitHub & Contributing to open-Source Project**
3. **Creating your GitHub Portfolio Page & More Features to Explore**

Why we use Git Hub:

Four Core Reasons Why GitHub:

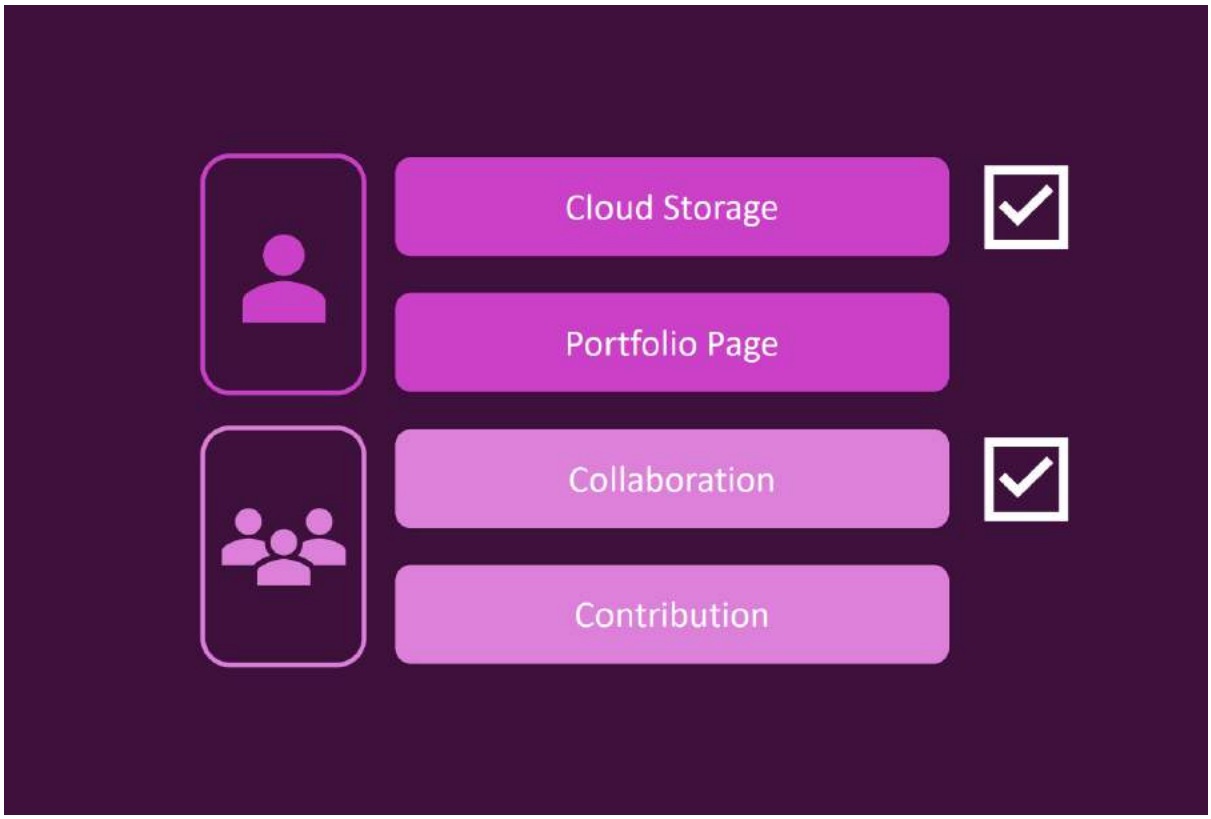


Fig. The 4 GitHub Use Cases

- **Cloud Storage: the single user uses GitHub, for example, to have a cloud storage of his or her own Git projects.**
- **Portfolio Page: The single user might also use GitHub to present a portfolio page, so a page presenting all the core skills, all the great projects he or she worked on, or is working on.**
- **Collaboration: Collaboration GitHub means that either you have a project, but you want to add other people, but to collaborate with you on this project**
- **Contribution: It builds your resume by demonstrating that you can collaborate with others on code.**

Understanding The Account Types:

Fig. Different Accounts in GitHub

Pricing Models official site: https://github.com/pricing

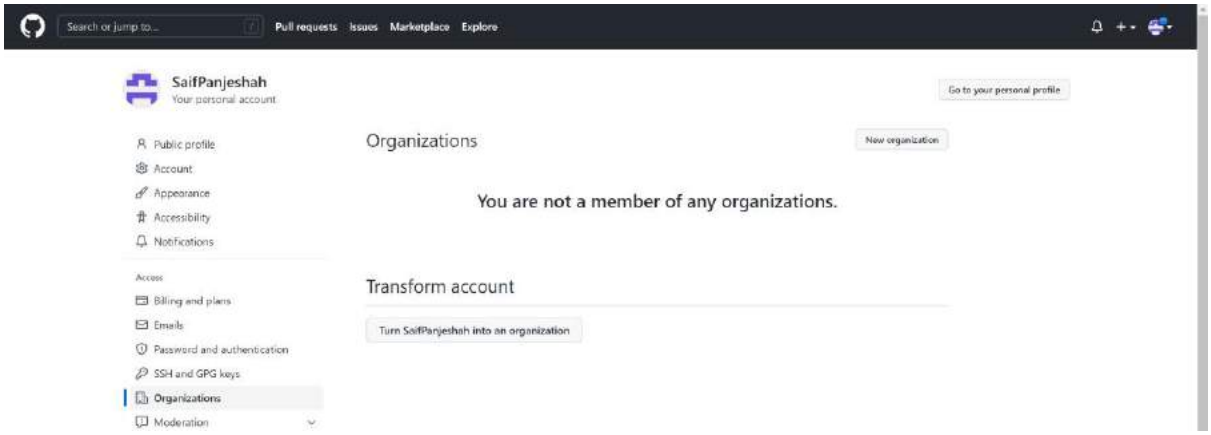


Fig. Personal GitHub Account

Changing Repository Type from Public to Private:

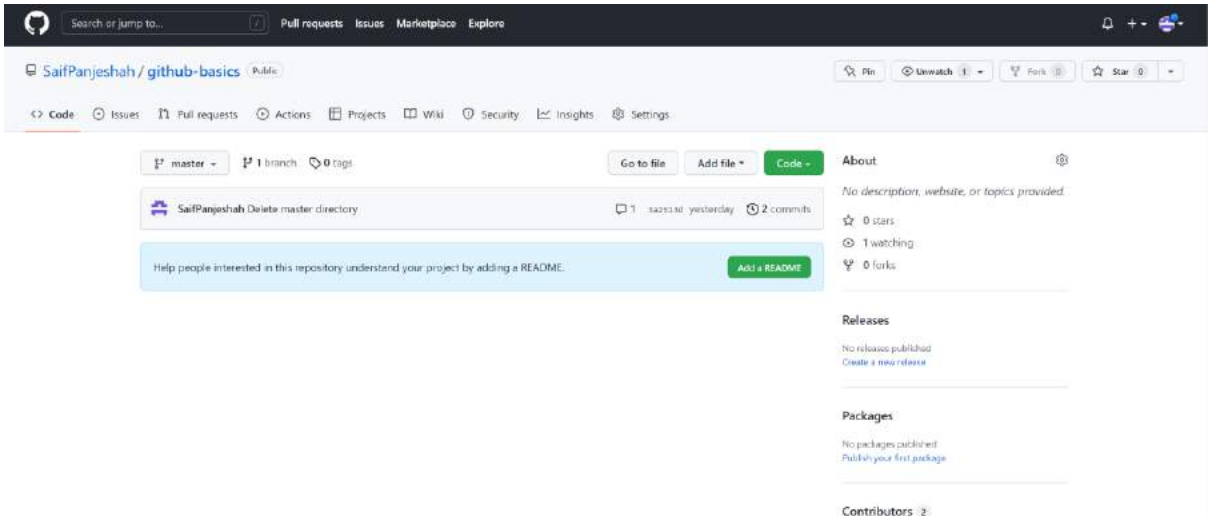


Fig. Shows Repository added in our account

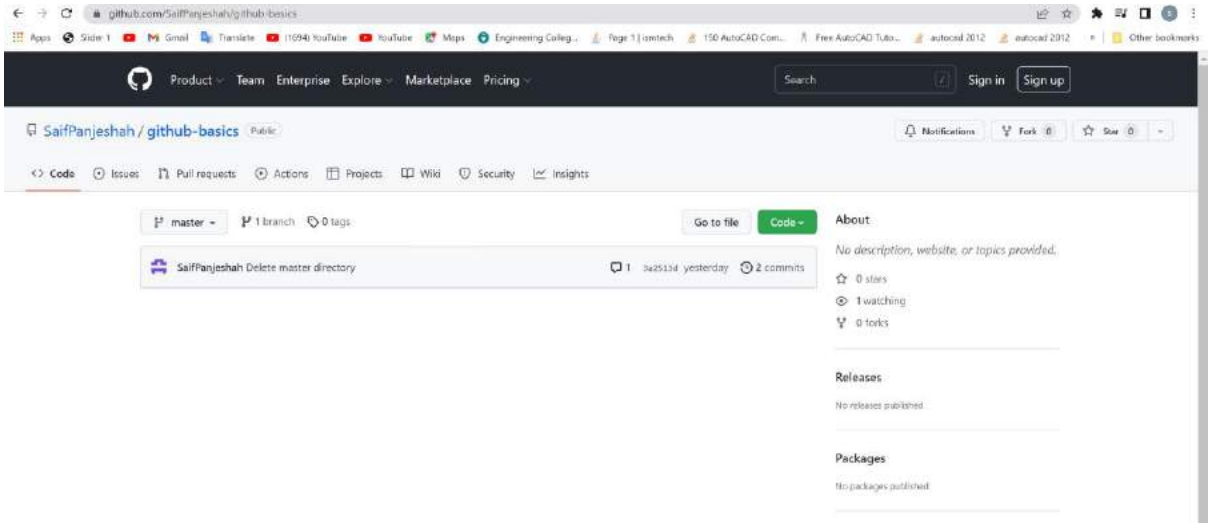


Fig. Accessing repositories from outside the browser or any user

How Can we avoid so other the users can't get the access?

Go to the Setting and Change the repository visibility in Danger Zone to private

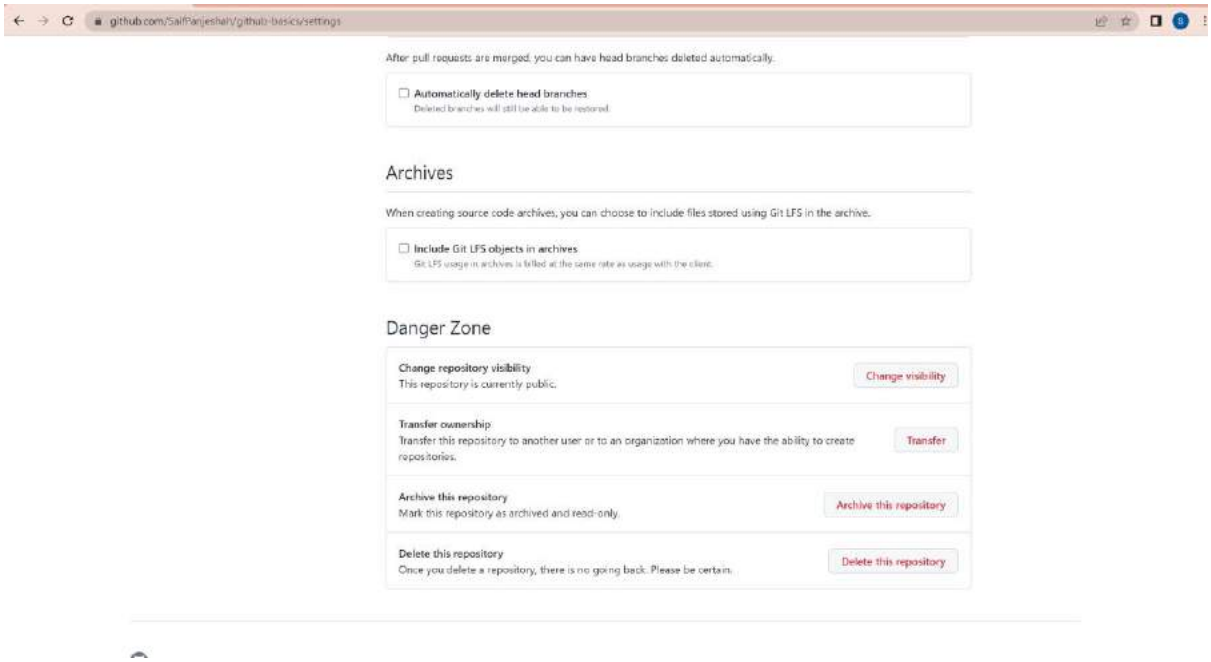
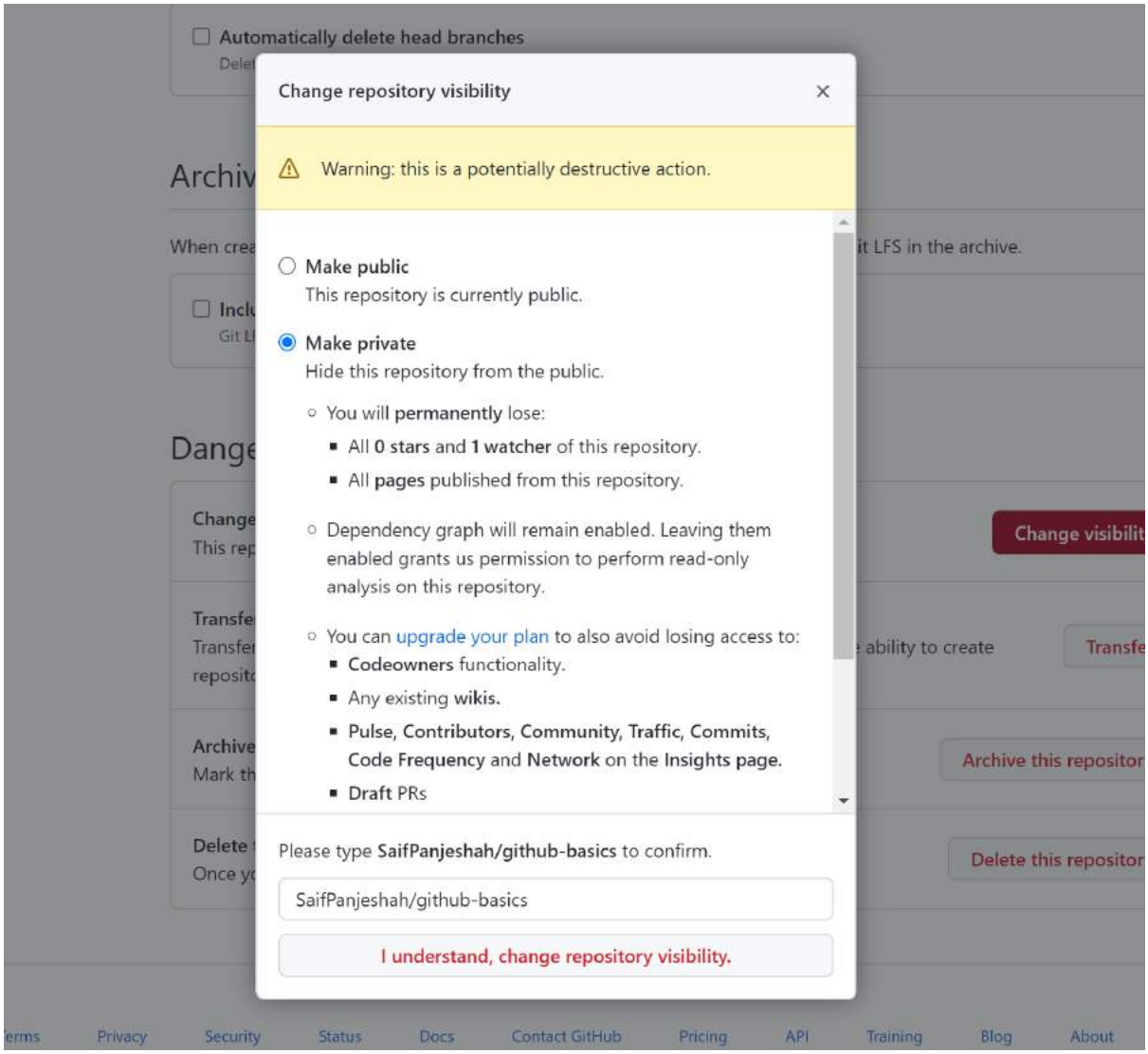


Fig. Visibility Setting of repositories



Confirm access

Password

••••••••••••

Forgot password?

Confirm password

Tip: You are entering sudo mode. We won't ask for your password again for a few hours.

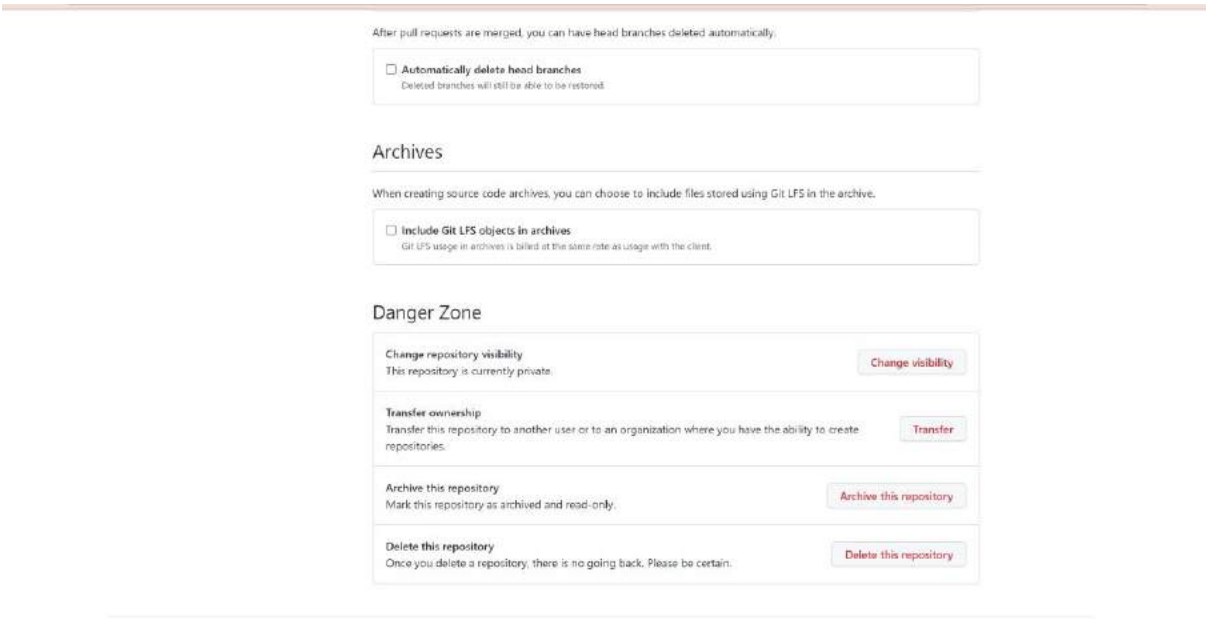


Fig. Successful Changes from public to private repository

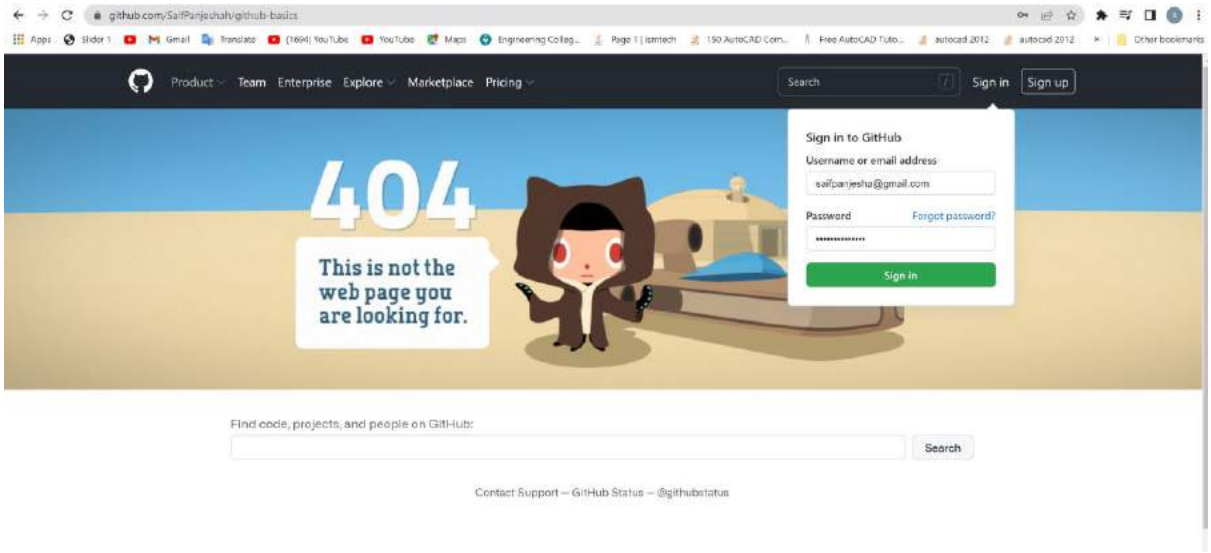


Fig. No one can access the repository

Remember: For Changing the repository public follow the Steps Vice Versa.

Pushing Commits to Public Repository

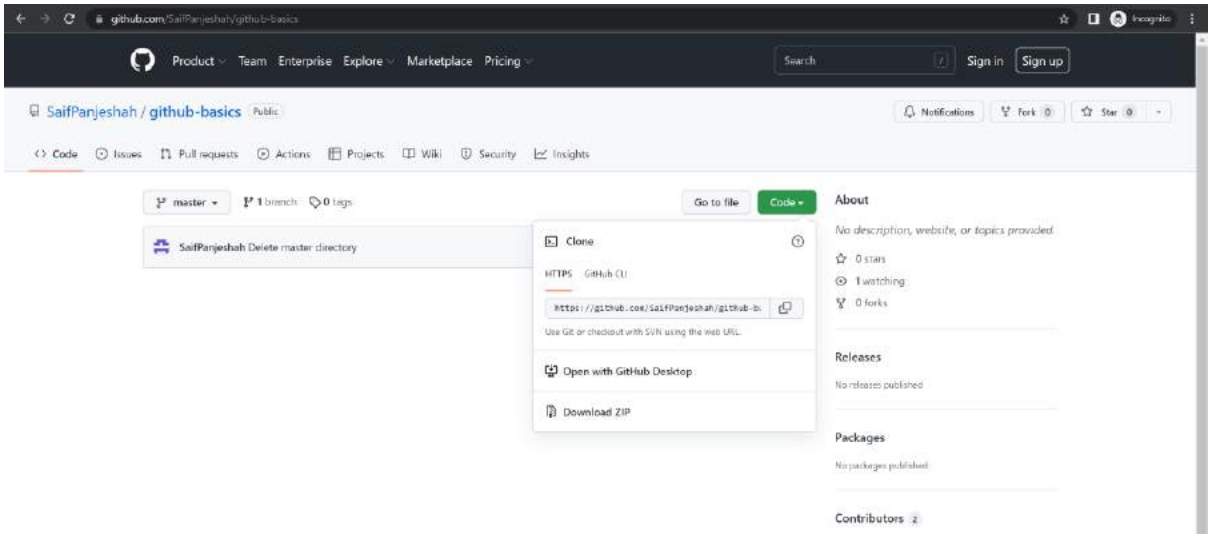


Fig. Changing the repository to public

Copying the Http URL: https://github.com/SaifPanjeshah/github-basics.git **and create new folder in vs code push the information our actual GitHub Project**

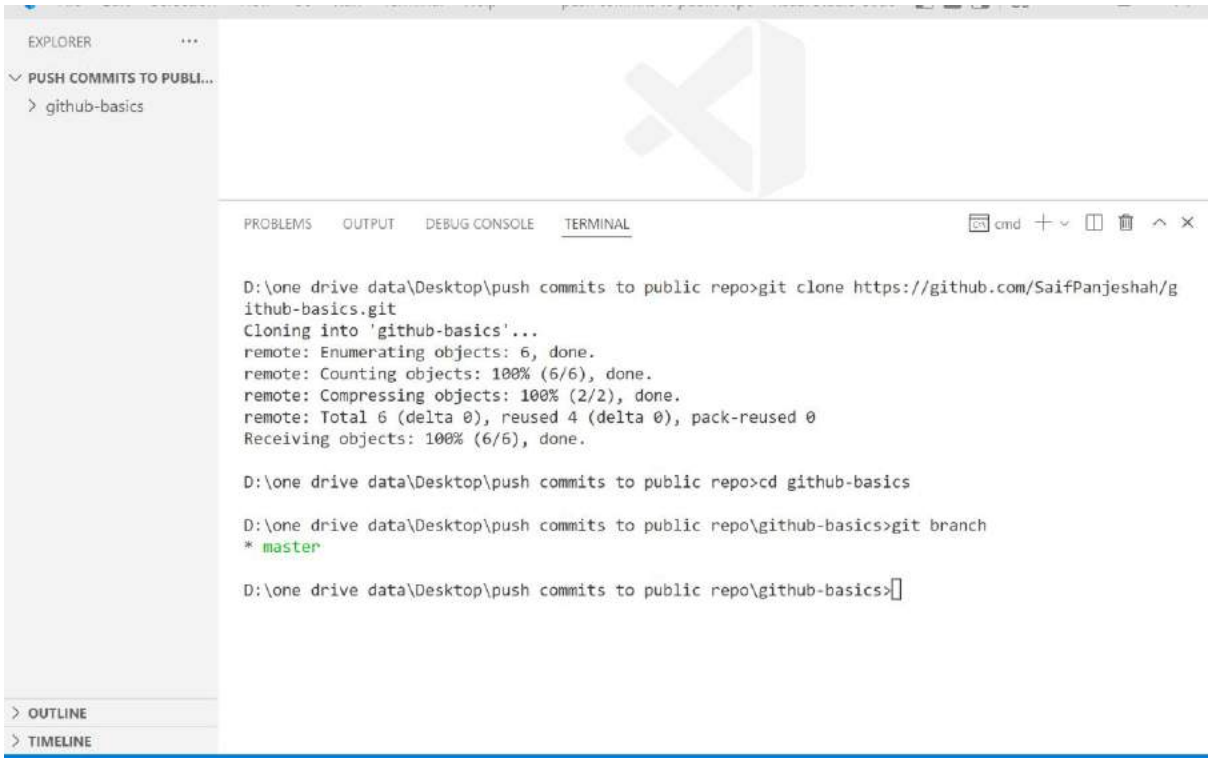


Fig. Created new push commits to public folder

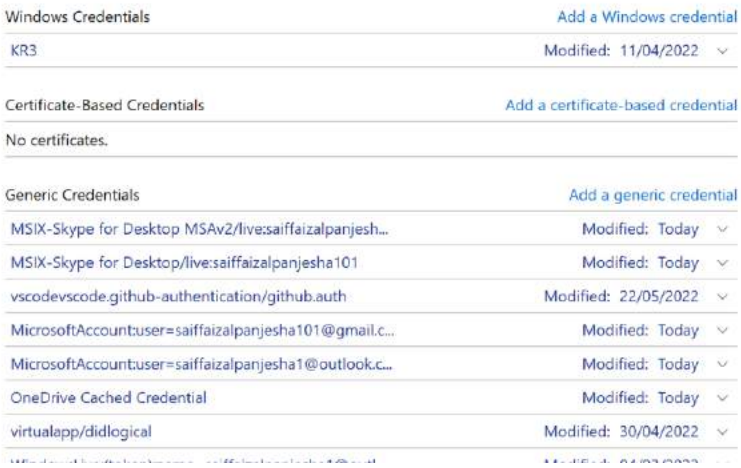


Fig. deleting all GitHub credentials

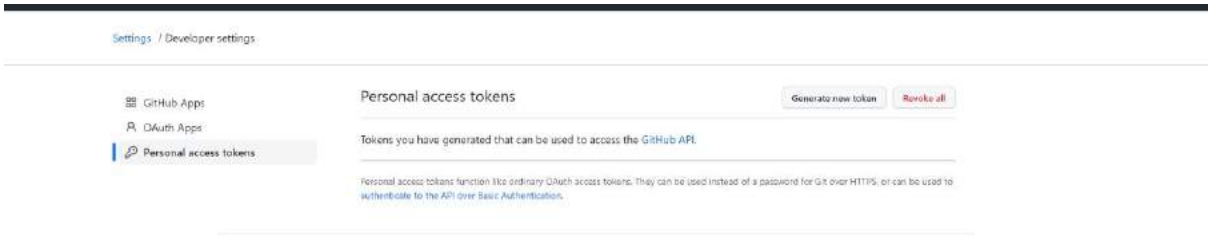


Fig. deleted personal access tokens

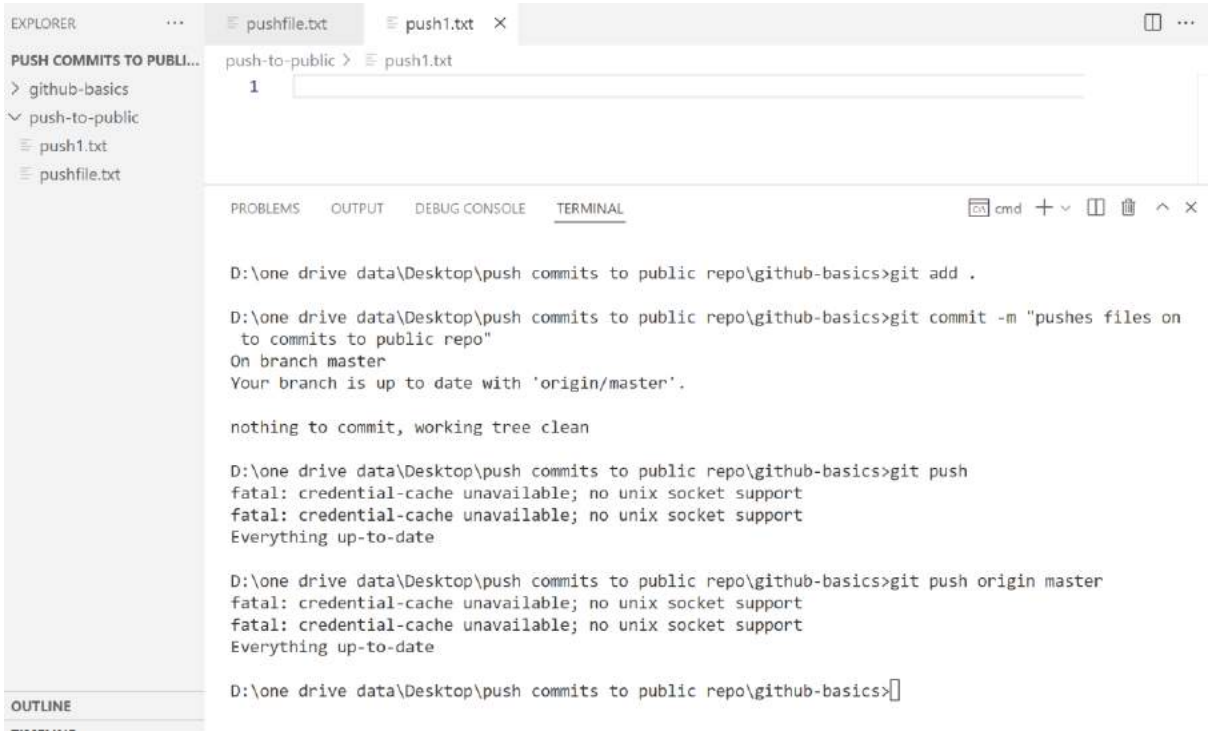


Fig. Shows with access other users can’t push the information our actual GitHub

How GitHub Account Manager Security:

Fig. Added Account Manager Security

Understanding & Adding Collaborator to a Private User Accounts:

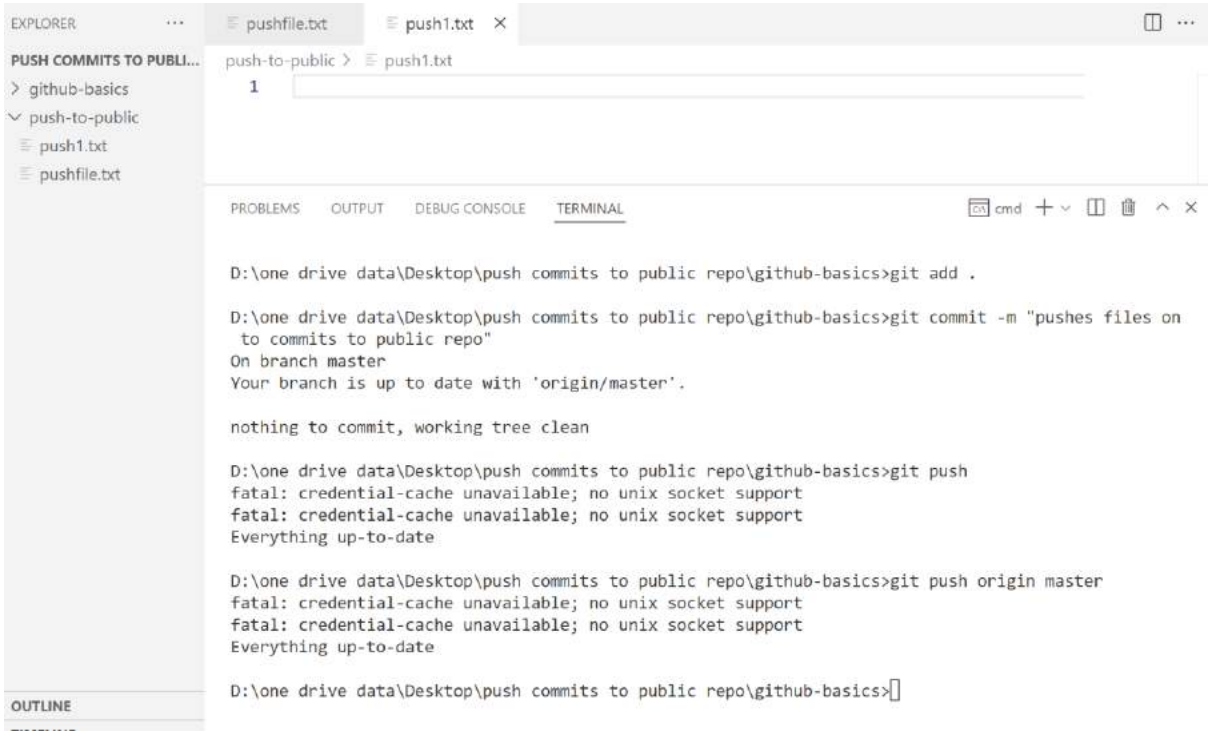


Fig. Shows with access other users can't push the information our actual GitHub

To resolve this problem:

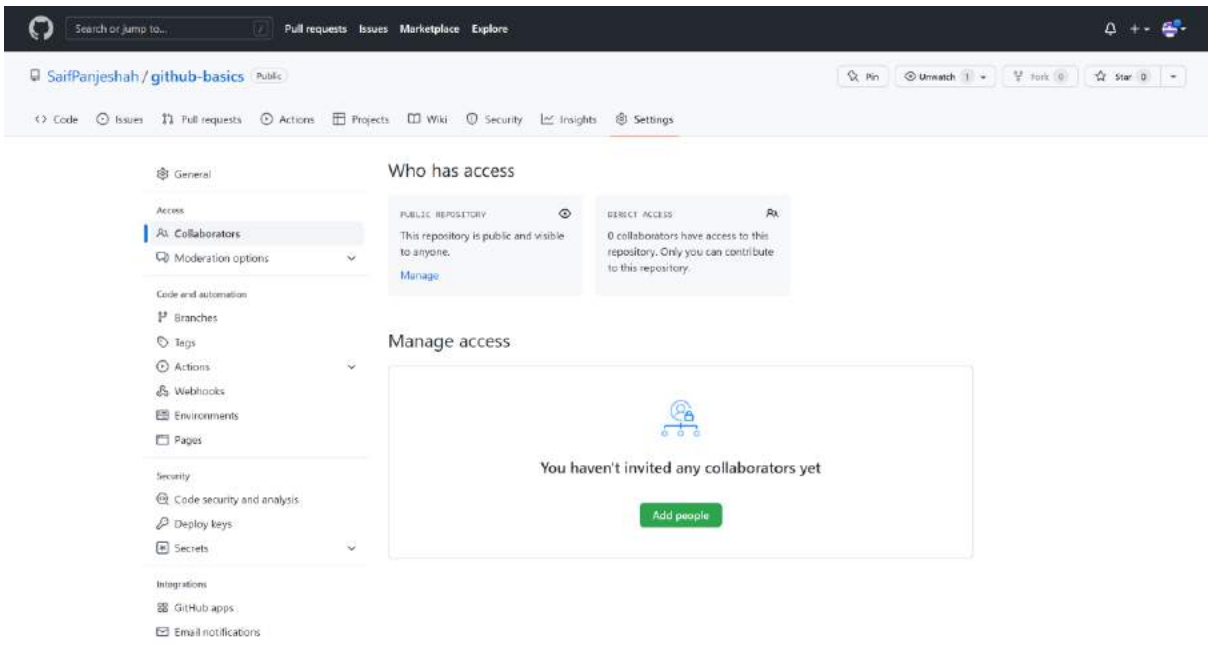


Fig. Adding Collaborator to a Private User Accounts

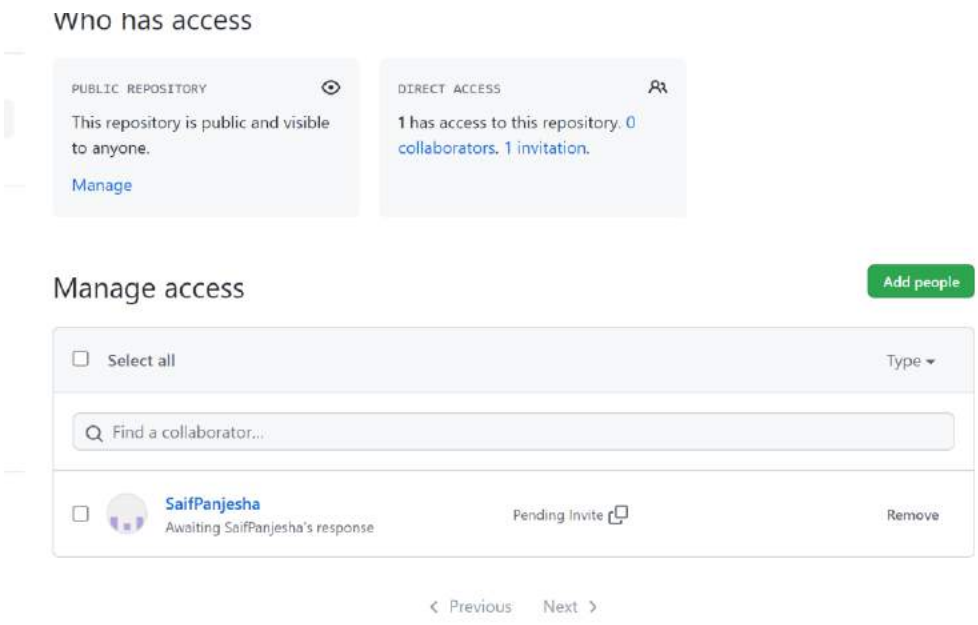


Fig. Pending User Invitation

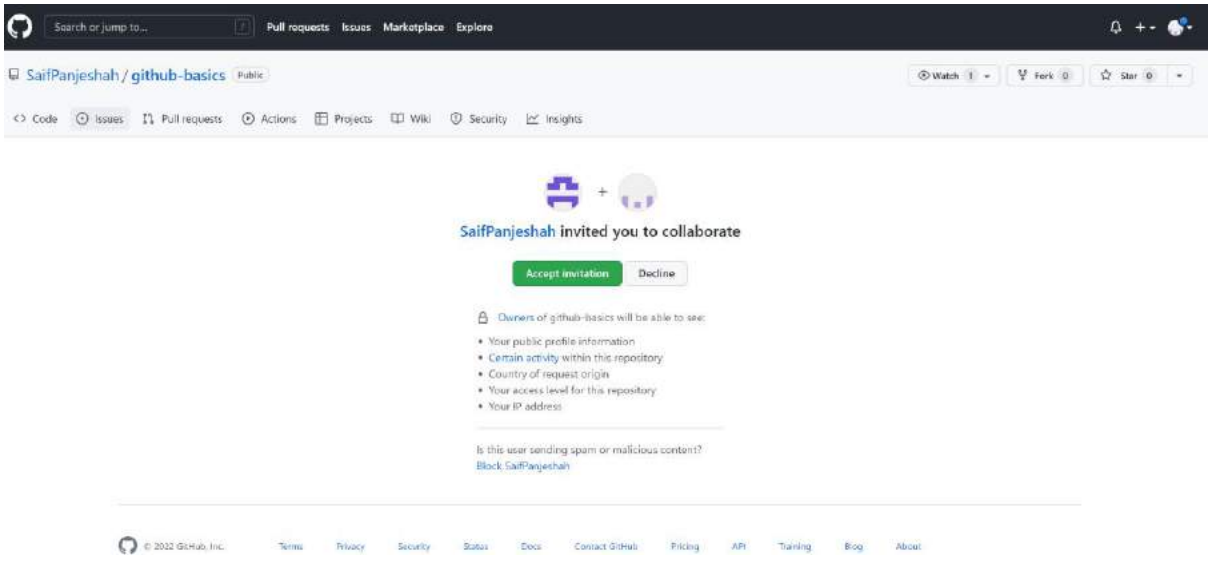


Fig. Accepting User request from another user account

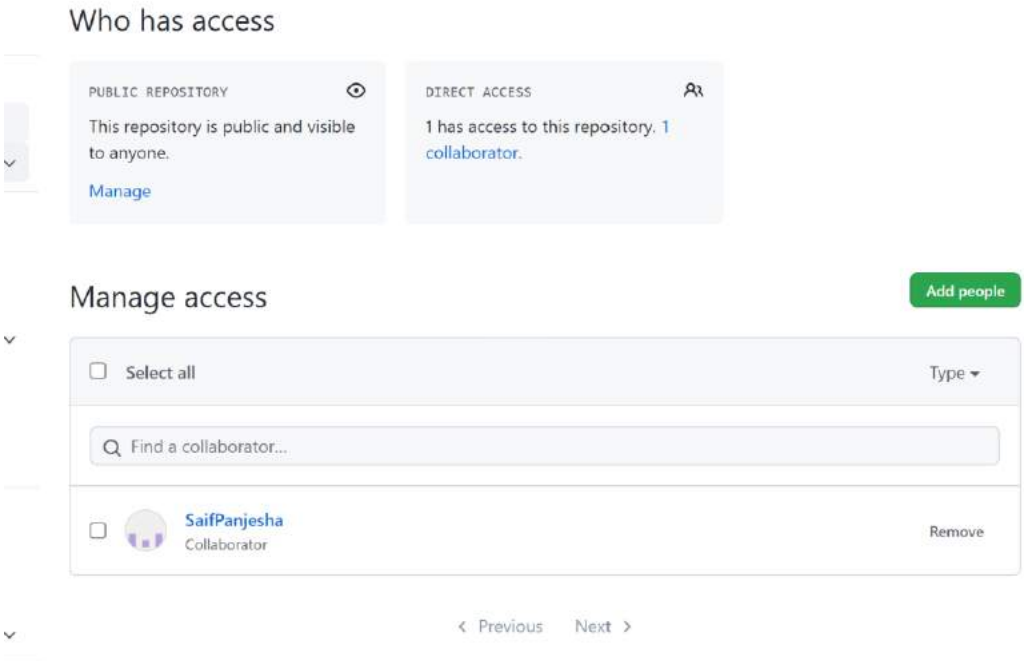


Fig. Successful added

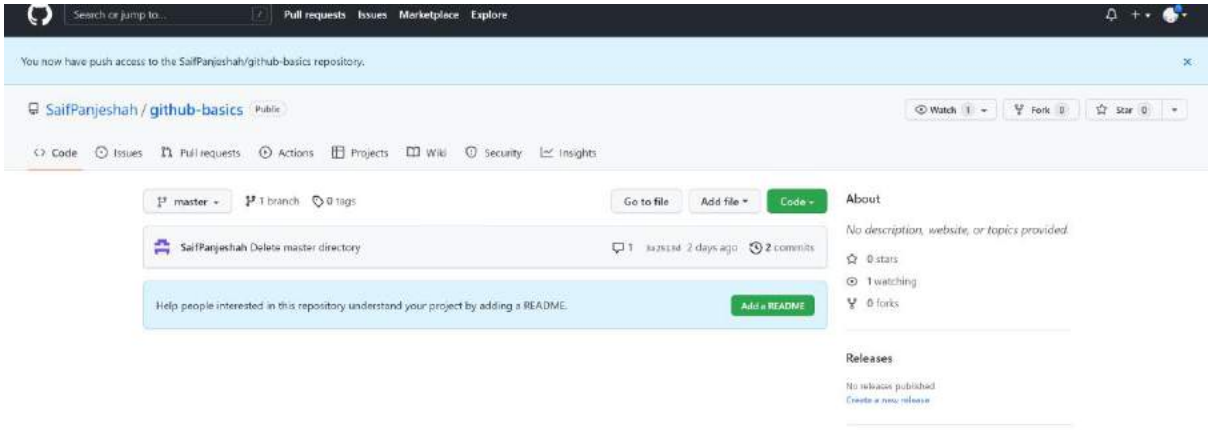


Fig. push access to another users

Now, here the problem is even with access users can't push the information our actual GitHub. because of personal access token and we as account owner can't share the personal access token.

Remember: It's better to get personal access token requests from another users (Collaborators)

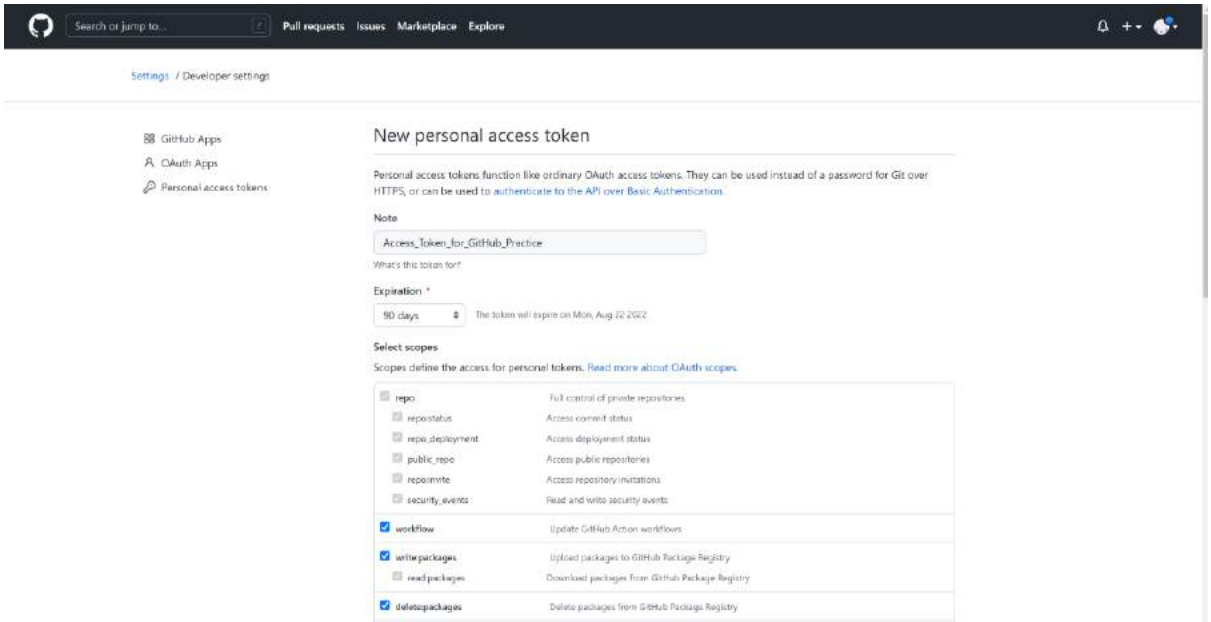


Fig. creating access Token From another user accounts

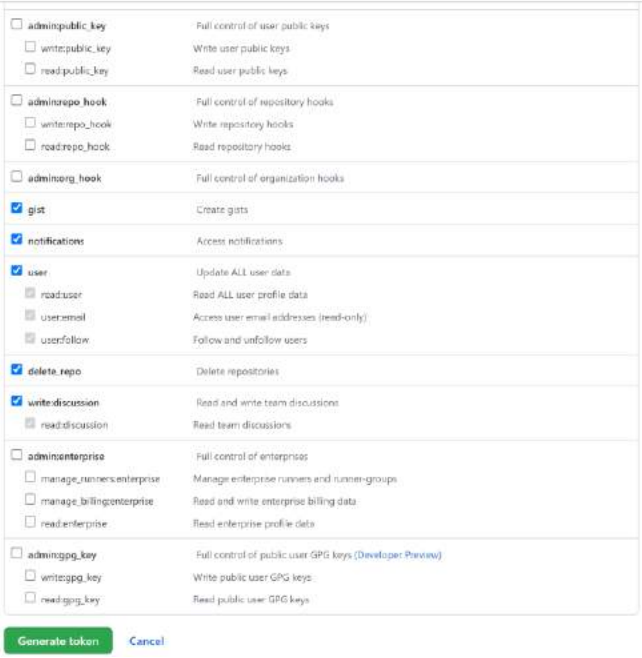


Fig. Creation Successful generate the token now without admin access

Created new file1.txt from local tracking branch of another user

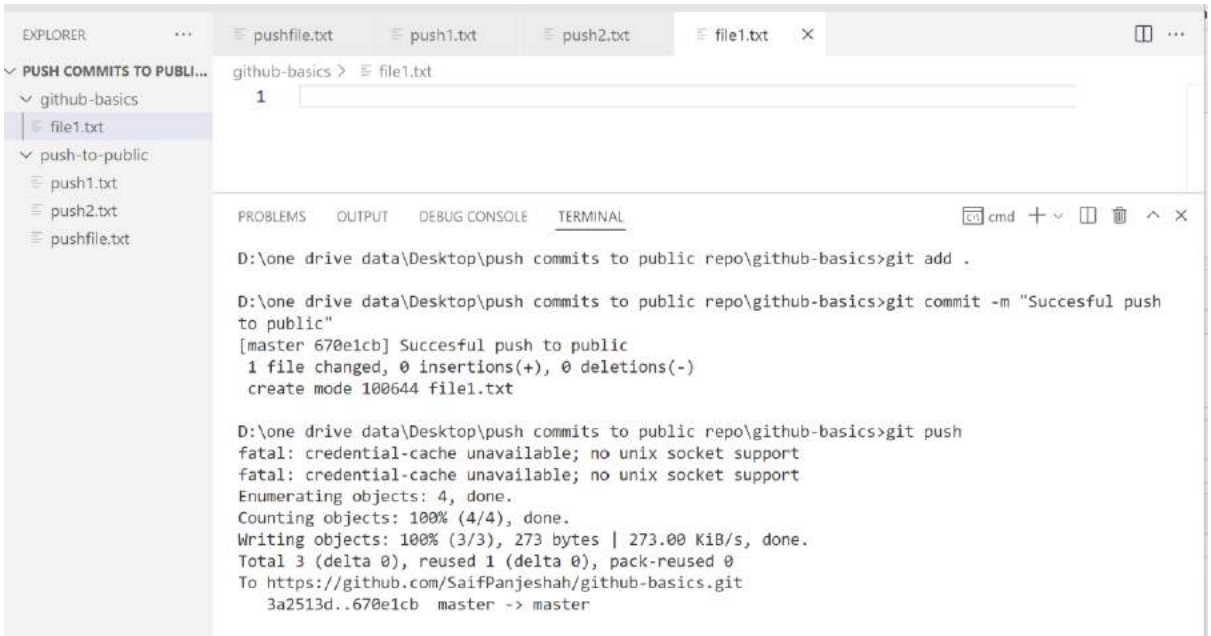


Fig. Succesful push the changes in account

To view please check the owner GitHub account

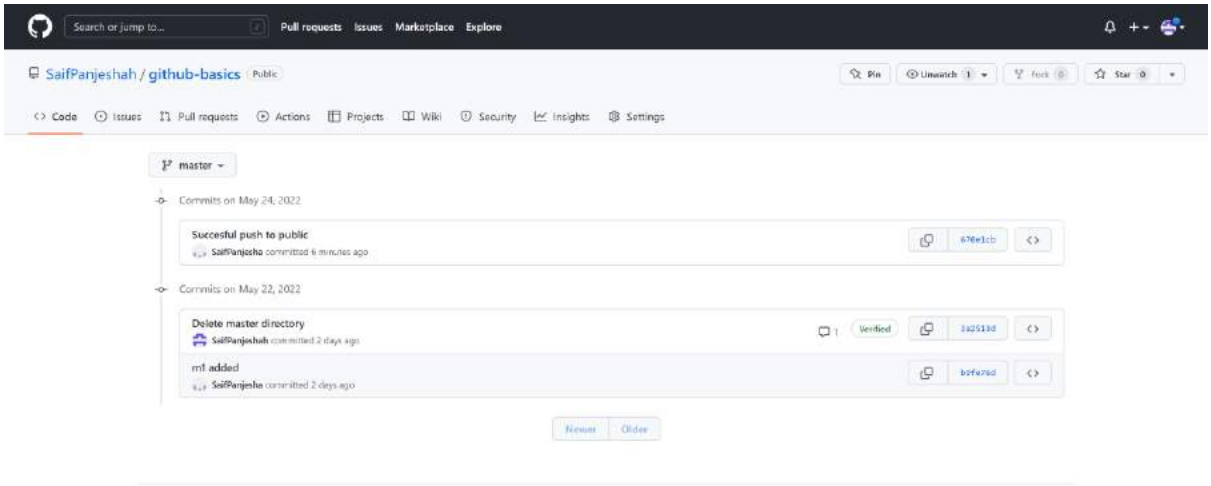


Fig. Commit Succesful added

Collaborating in Private repositories:

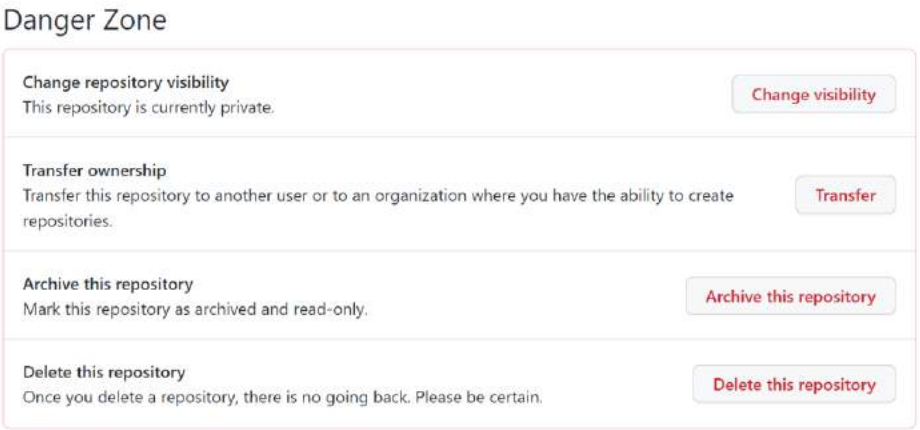


Fig. Changing repositories in private

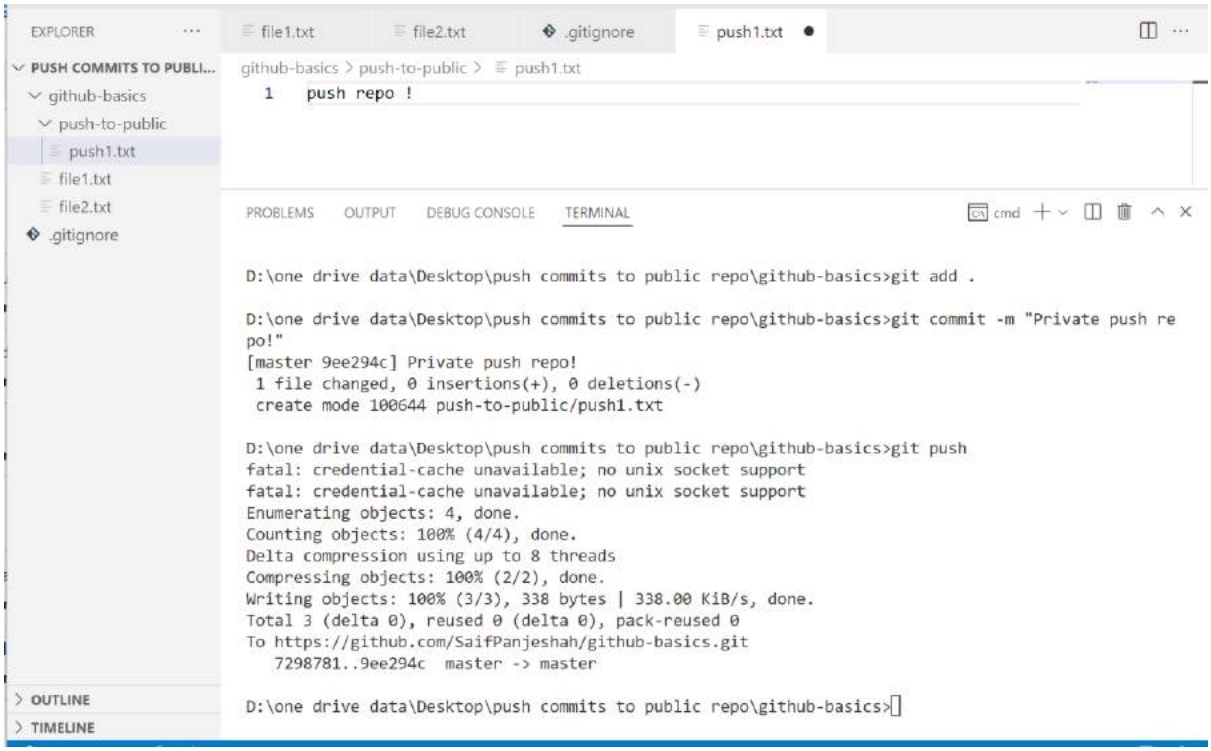


Fig. Successful push to private repository

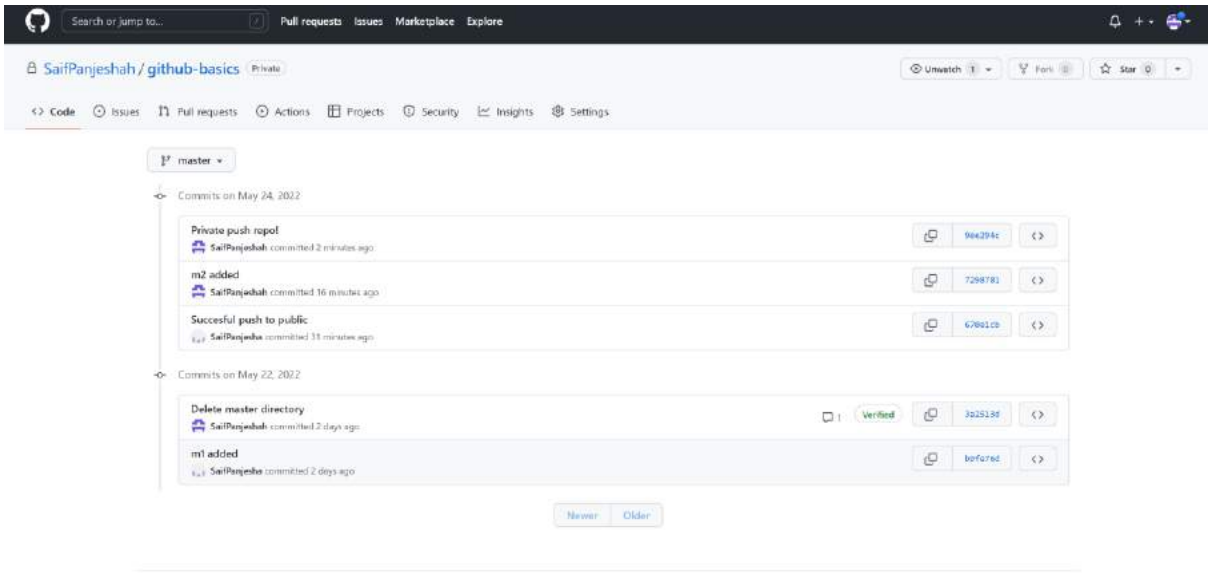


Fig. Succesful push to private repository in GitHub

Comparing Owner & Collaborator Rights:

Official Site:

https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-personal-account-settings/permission-levels-for-a-personal-account-repository

Limit Interactions:

Providing restrictions to another user.

Path: Goto Profile -> Settings -> Moderation

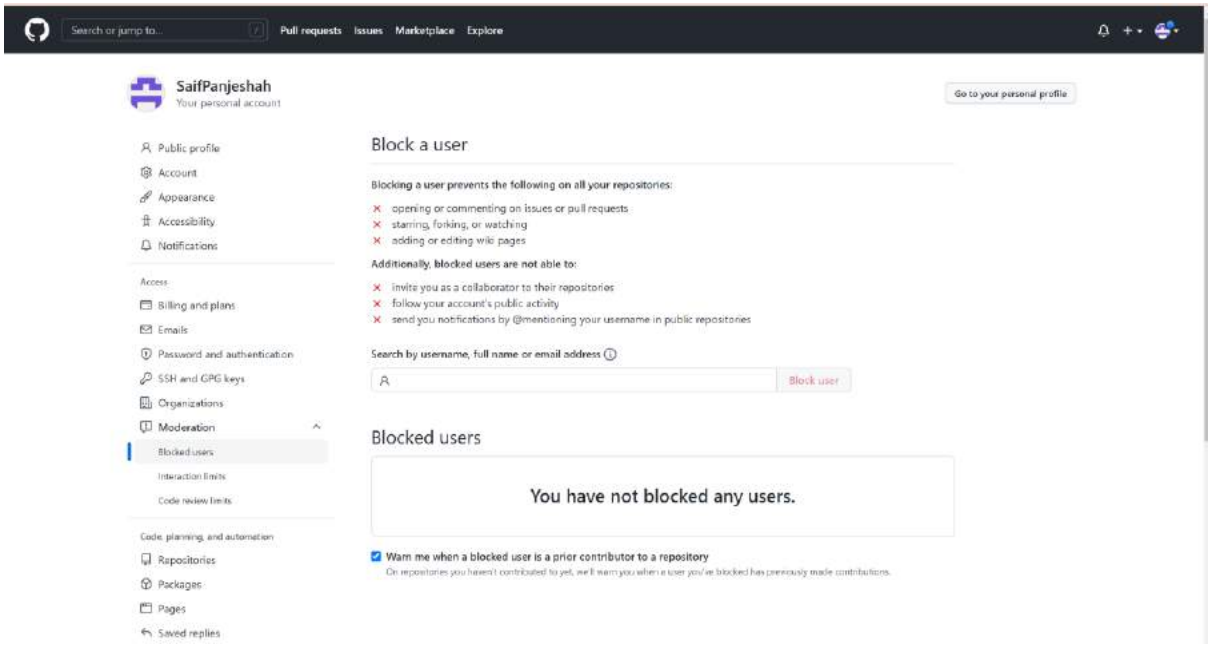


Fig. Block a user's

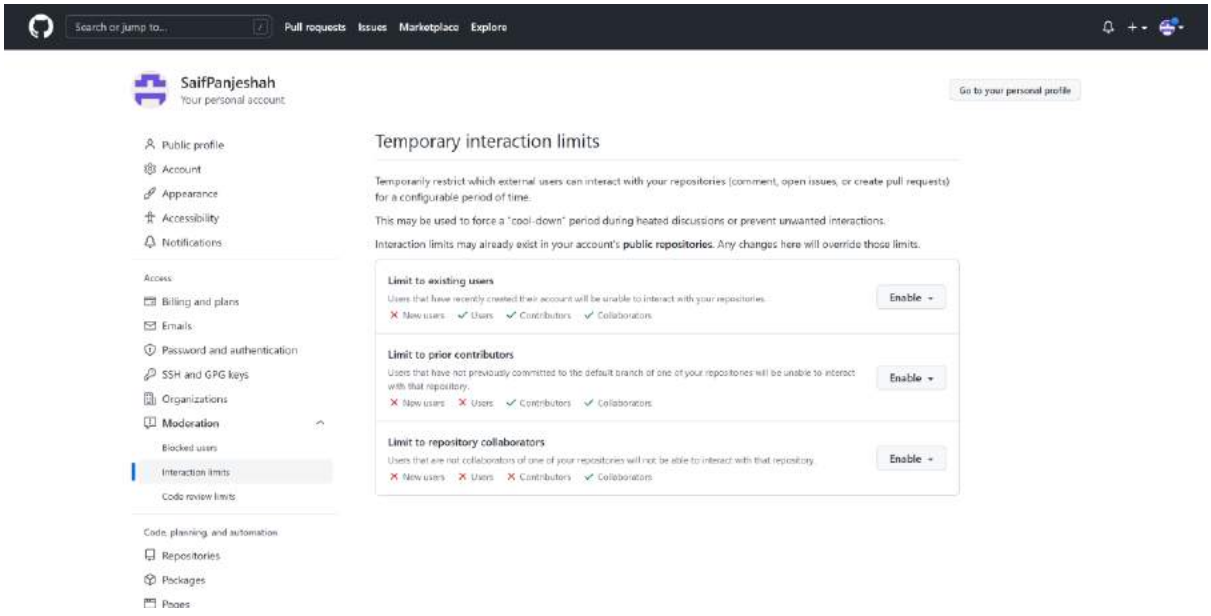


Fig. Limit Iteration

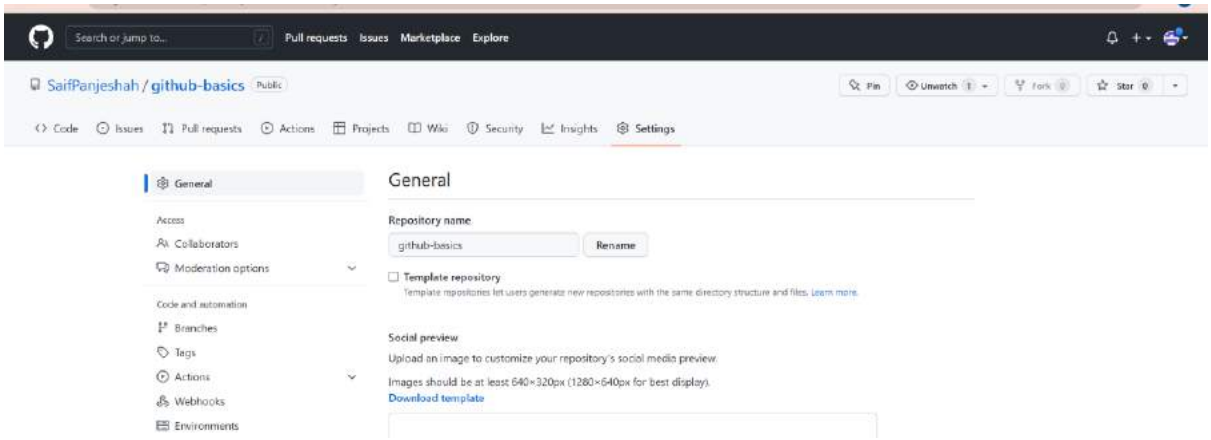


Fig. Changes repositories to public for applying Limit Iteration

Note : We cannot apply Limit to specific user only only on general conditions that apply to certain users.

Code Review Limit: Restrict users who are permitted to approve or request changes on pull requests in this repository.

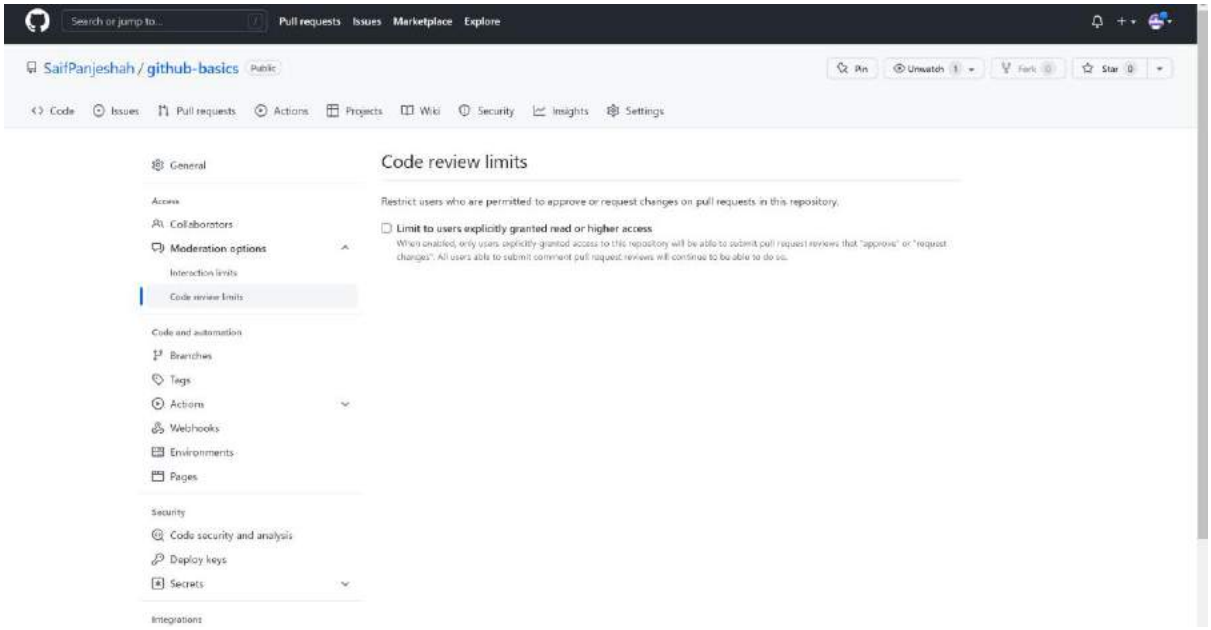


Fig. Code review limits

Introducing Organizations:

we have some limits here when it comes to managing specific access rights for collaborators of our projects

Fig. Added Account Manager Security

The big picture here of the whole security part. So, if you work in smaller projects with personal user accounts and some collaborators, you can perfectly do that. If you are part of a bigger project, or if you want to manage the specific rights the members of your project have, then an organization account might be the way to go.

Creating an Organization Account:

Go to Profile -> Setting -> Organizations -> Click on New Organization

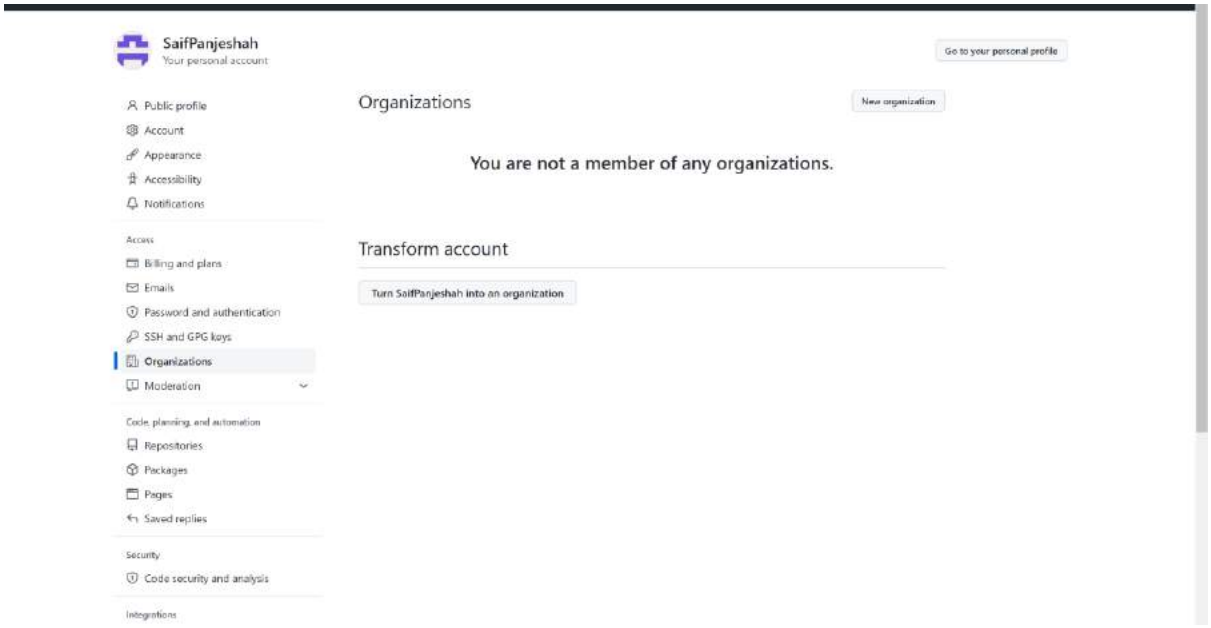


Fig. Creating organization account

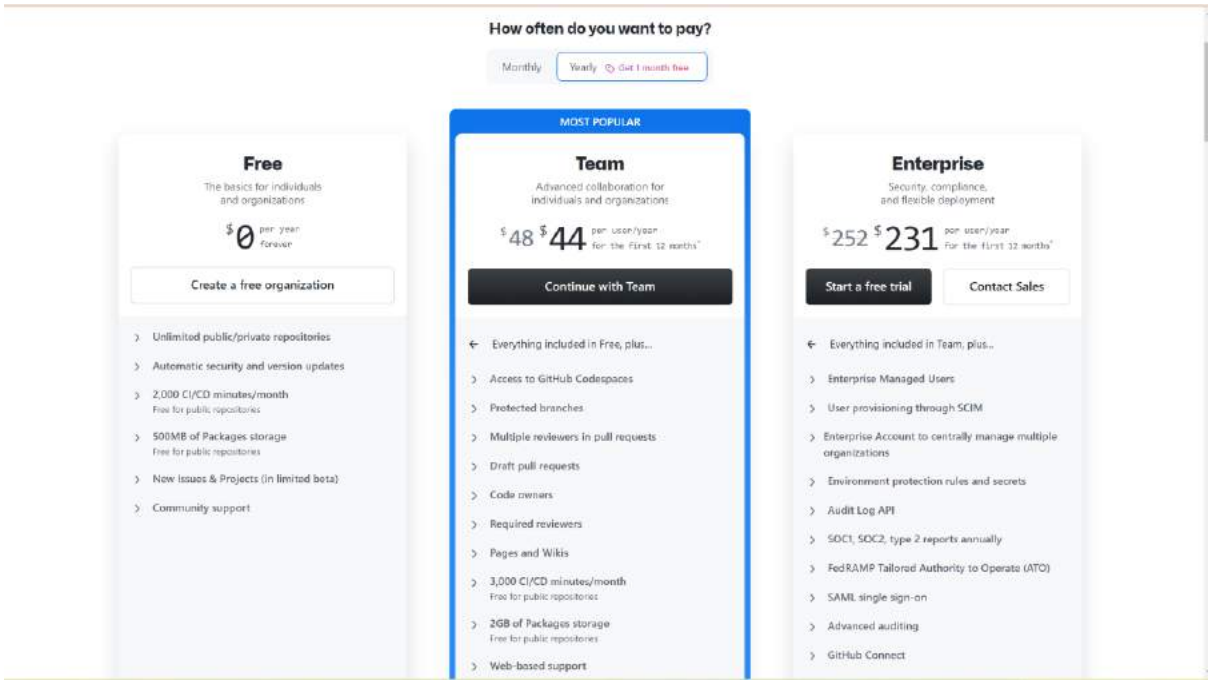


Fig. Pricing Start as Free Organization

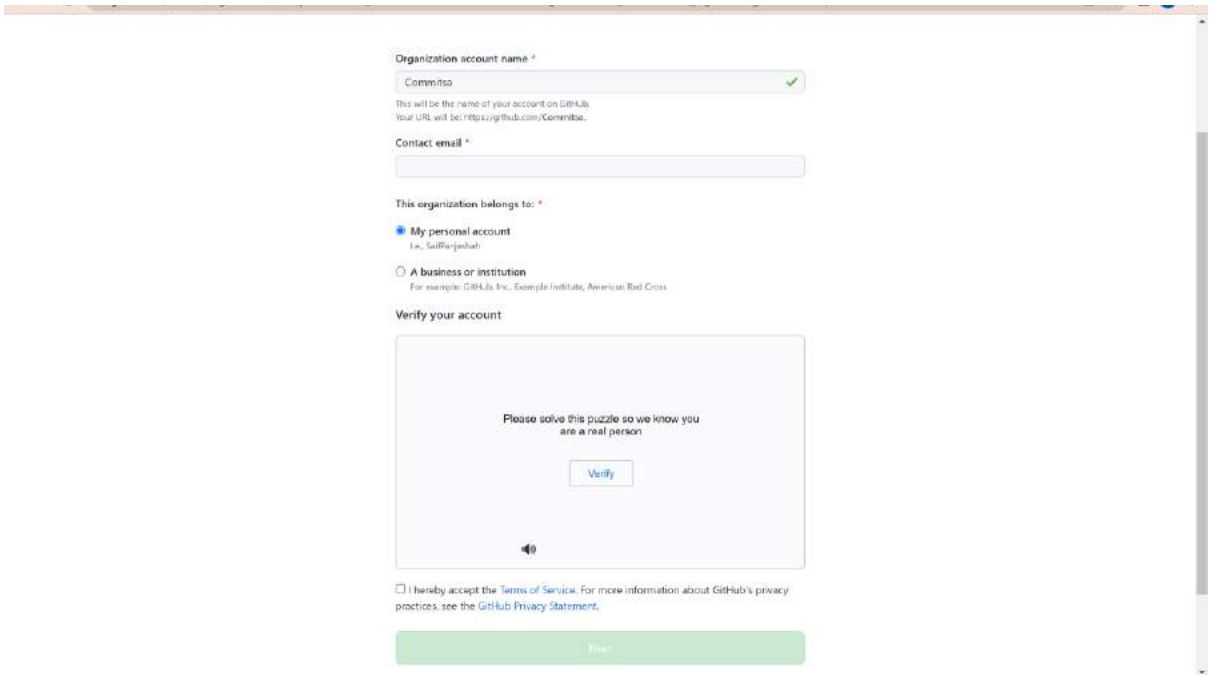


Fig. Adding Organization Name Commitsa

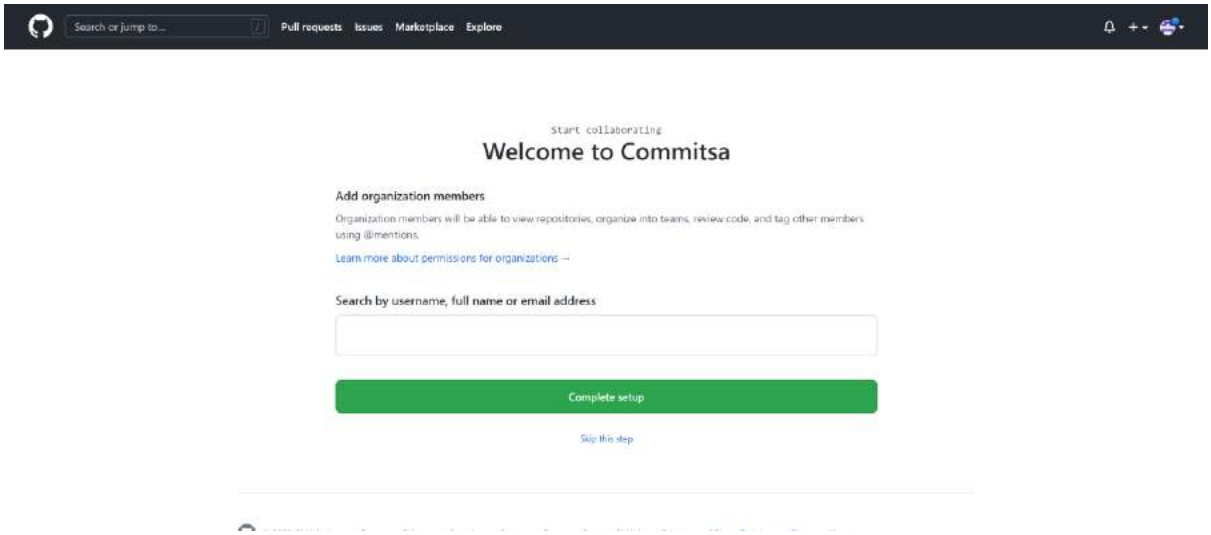


Fig. Verified Succesful

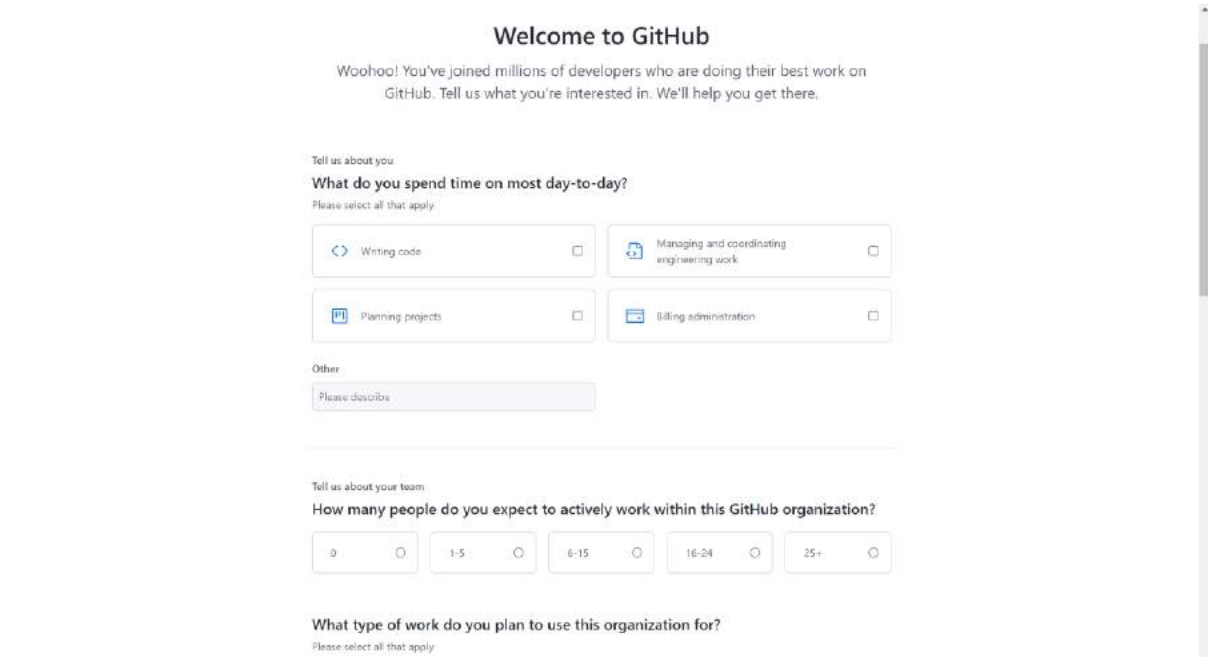


Fig. Complete the Setup and submit

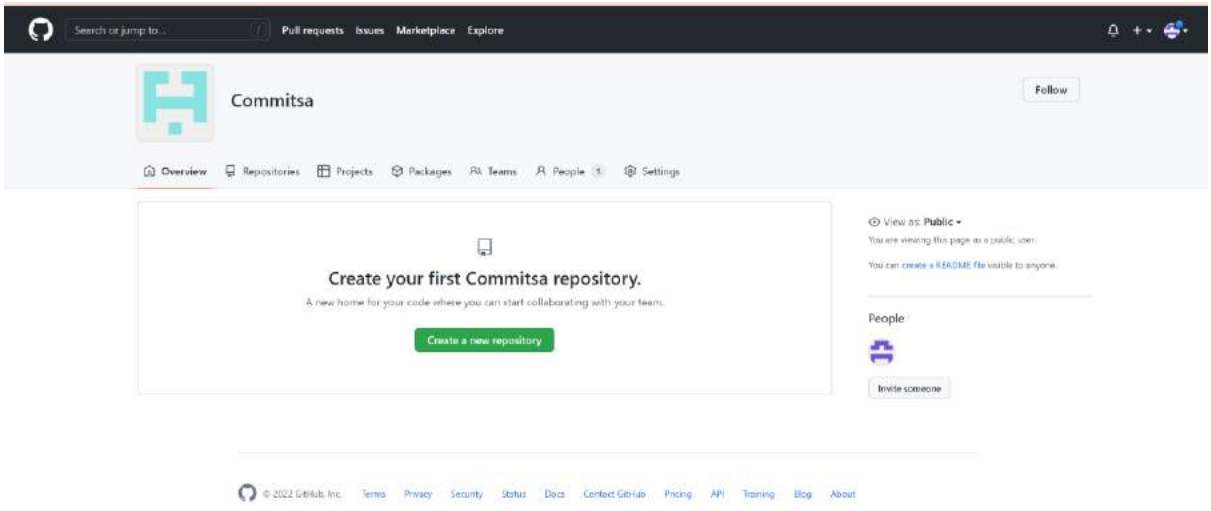


Fig. Successful Created

To View Organization from Dashboard

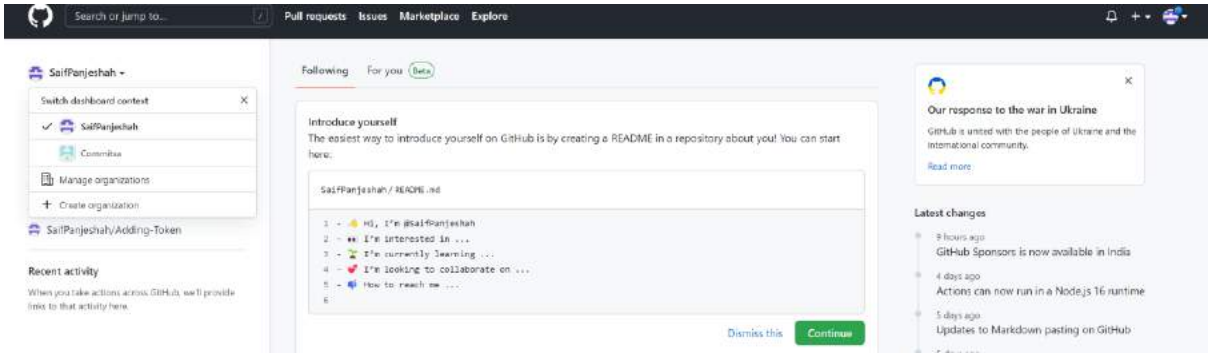


Fig. View Organization

Exploring Member Repository Permission:

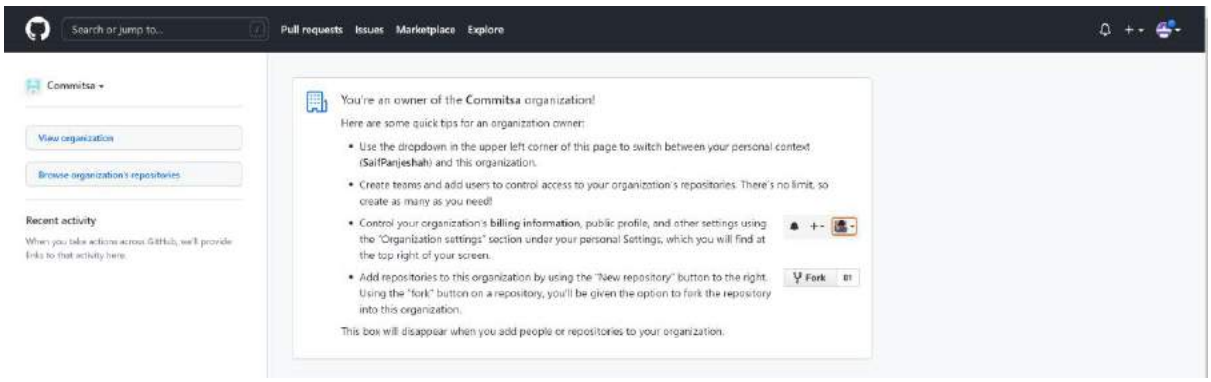


Fig. Commitsa Organization

What Missing, Here Member and repositories in Organization?

Creating New Repository:

Path: Inside Organization -> Create New Repo

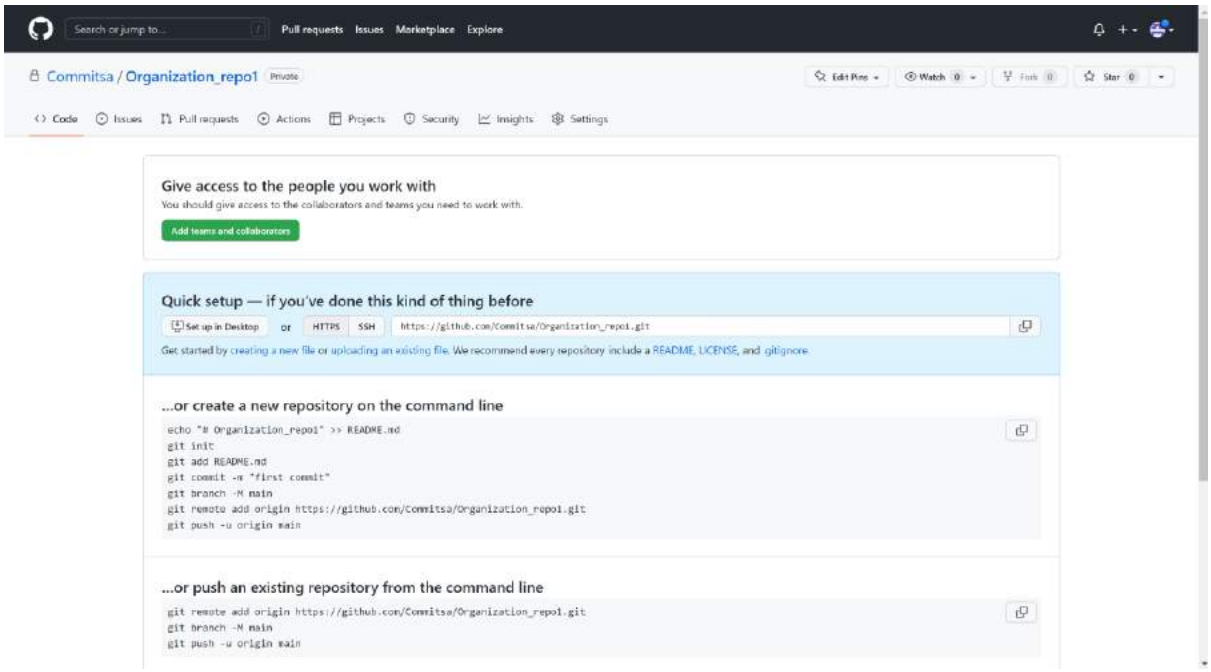


Fig. Successful Created Private Repository in Organization

After Creating repository Inviting Teams Members Same as Personal Account:

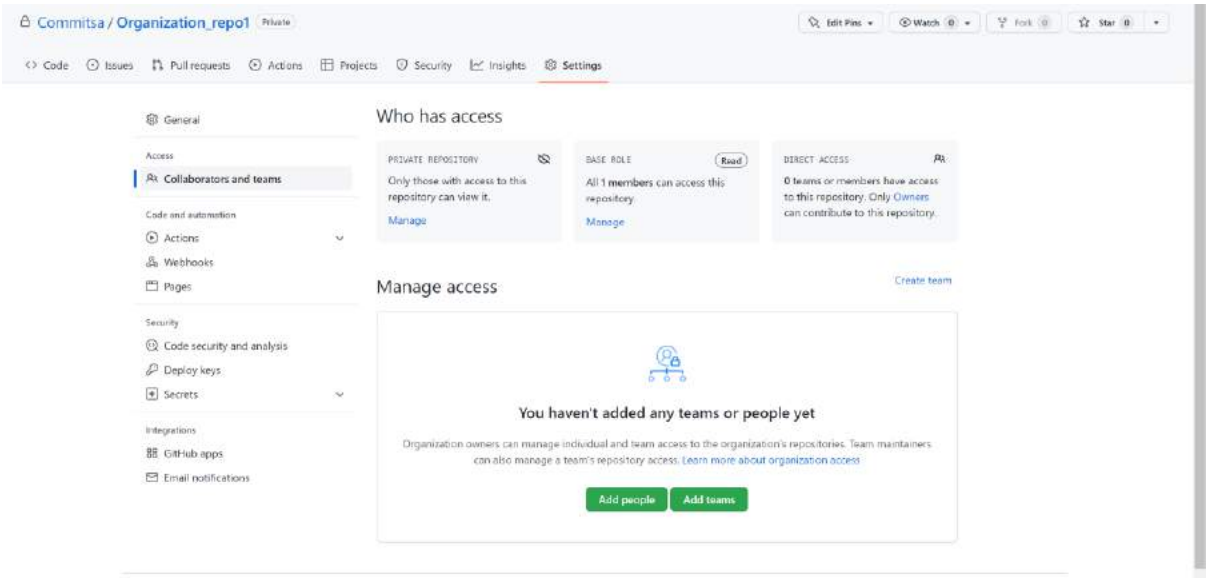


Fig. Manage access

We can privilege our base role member (account owner)

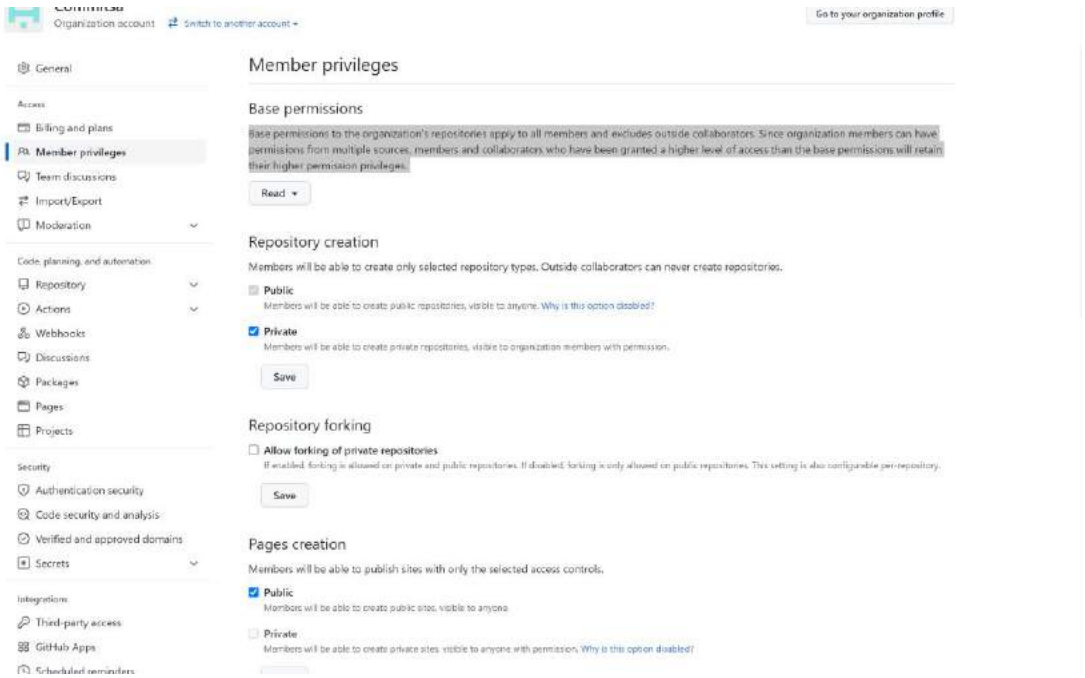


Fig. Base role privileges member

Adding Outside Collaborator:

An *outside collaborator* is a person who is not a member of your organization, but has access to one or more of your organization's repositories.

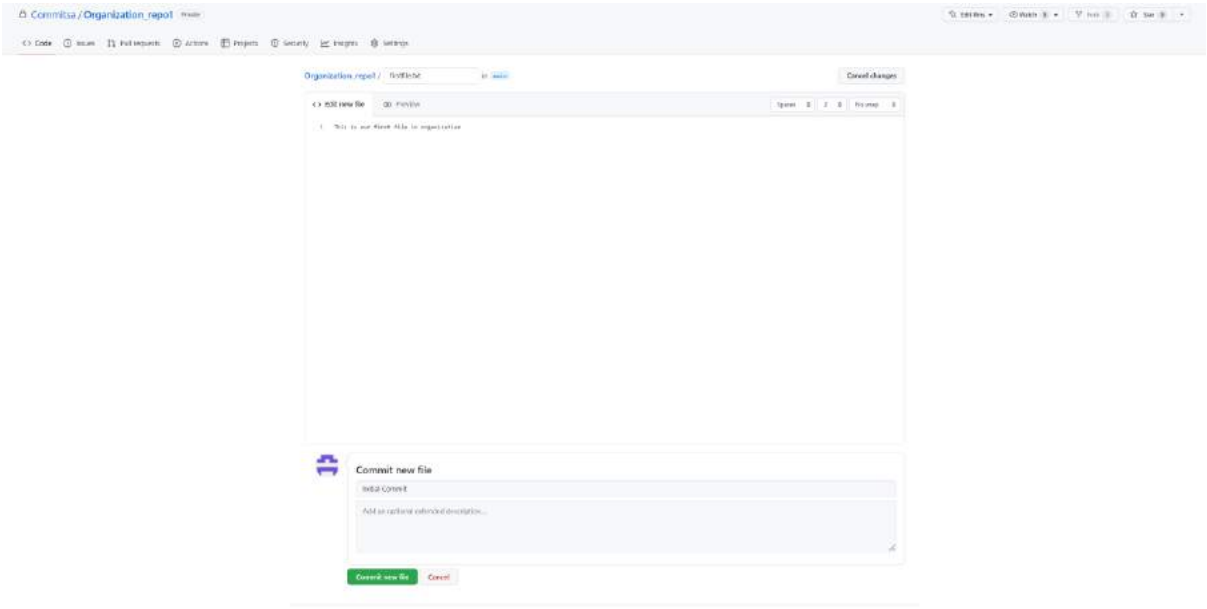


Fig. Adding First File in Organizational Account

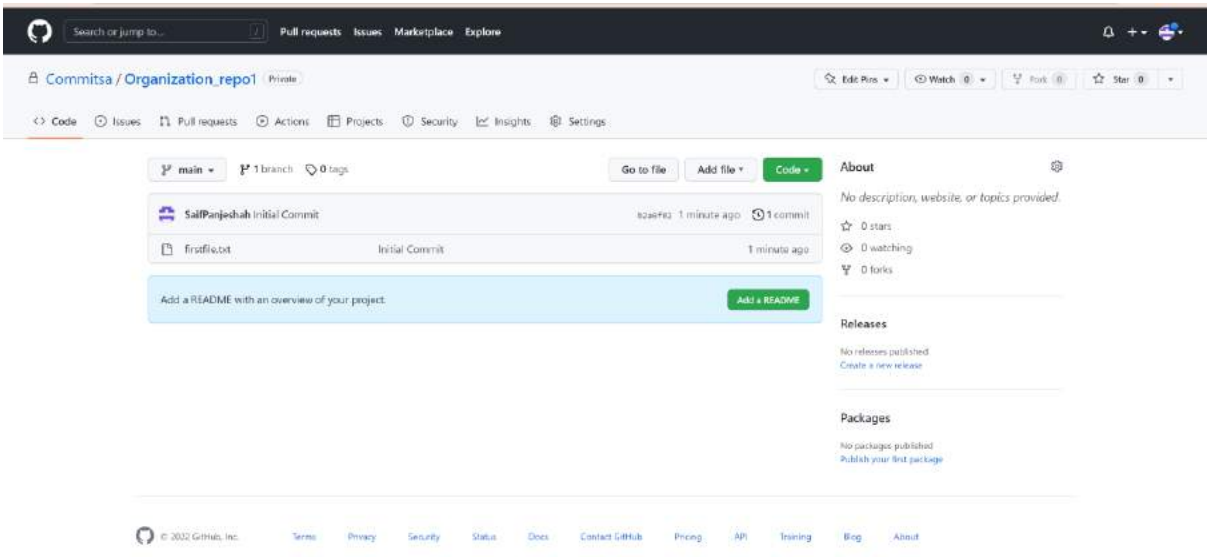


Fig. FirstFile.txt Created

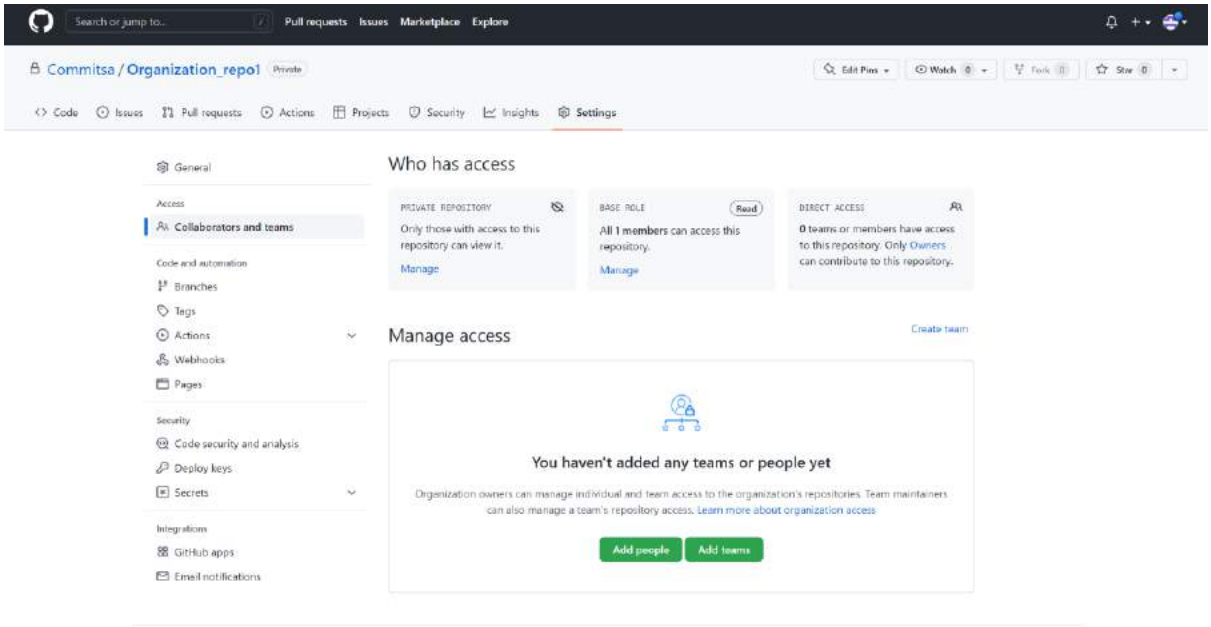


Fig. Collaborators and teams settings

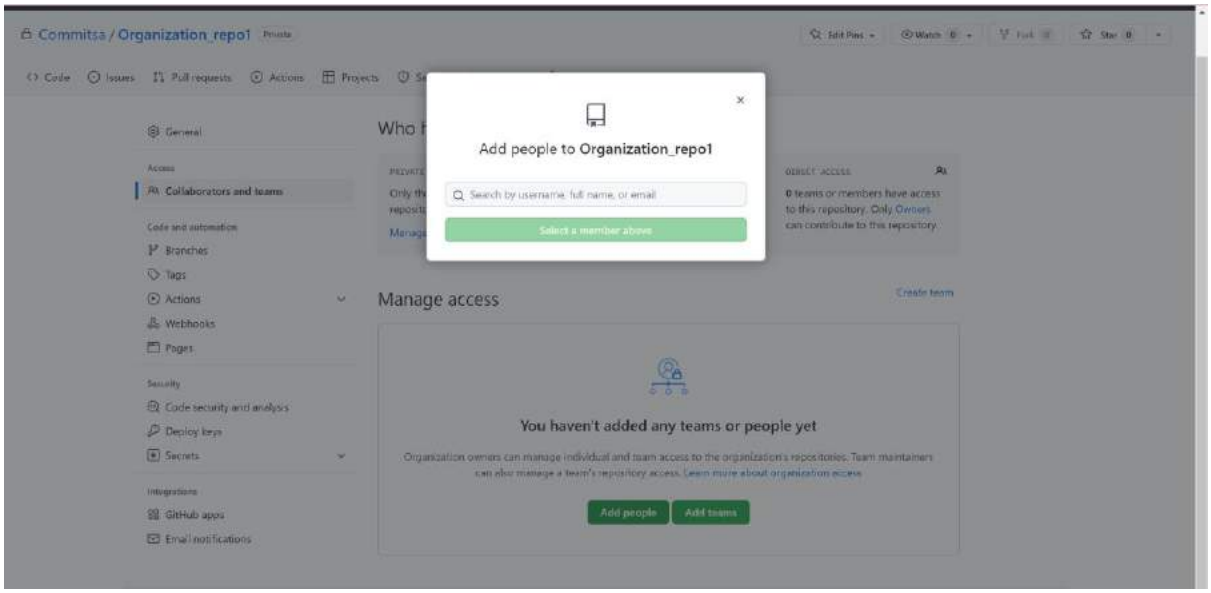


Fig. Adding Outside Collaborators Organization read only permission

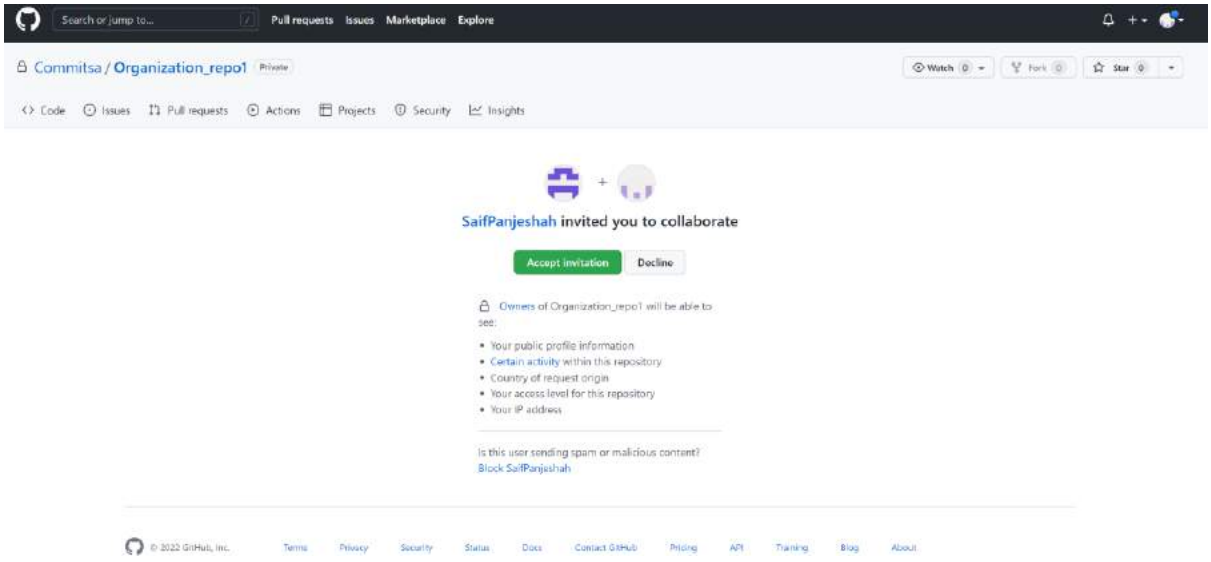


Fig. Outside Collaborators Organization Invitation

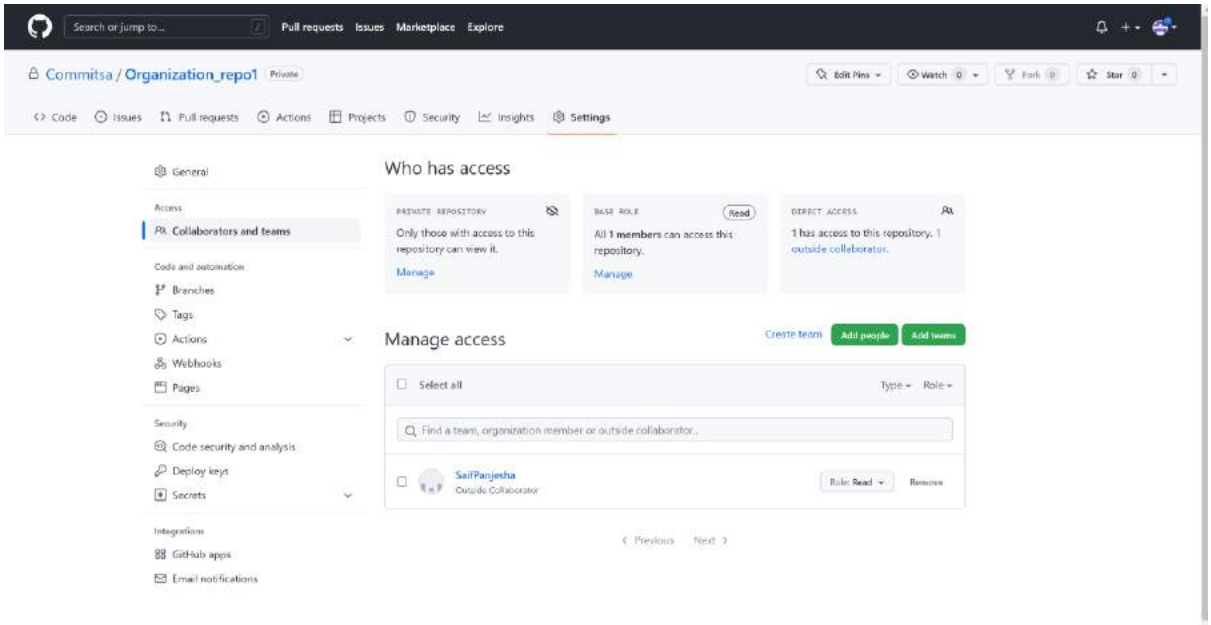


Fig. Successful added outside collaborator

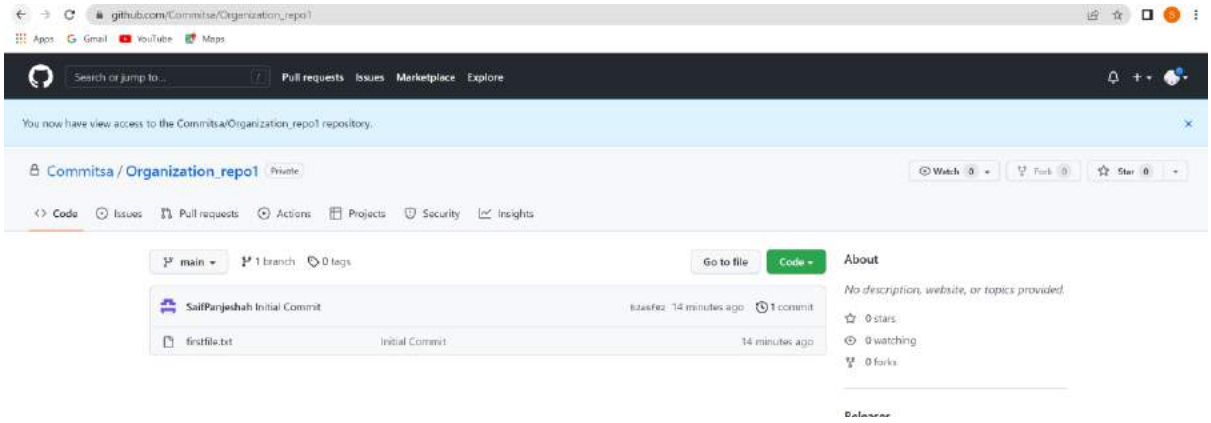


Fig. Given view access permission only

Now, trying to push the code from local branch git (VS Code) to check whether the code is successful to cloud or not.

git clone https://github.com/Commitsa/Organization_repo1.git .

Fig. Successful created organization files

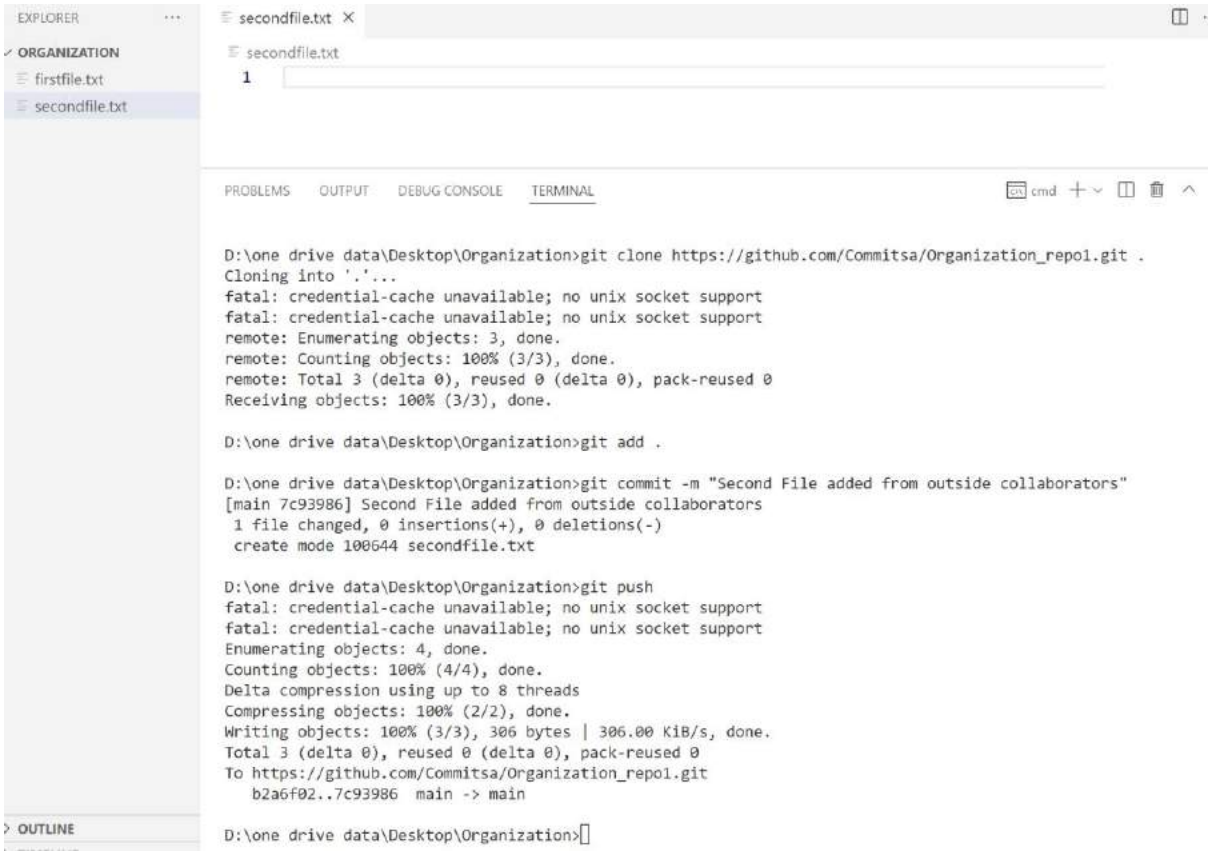


Fig. Successful added to organization file

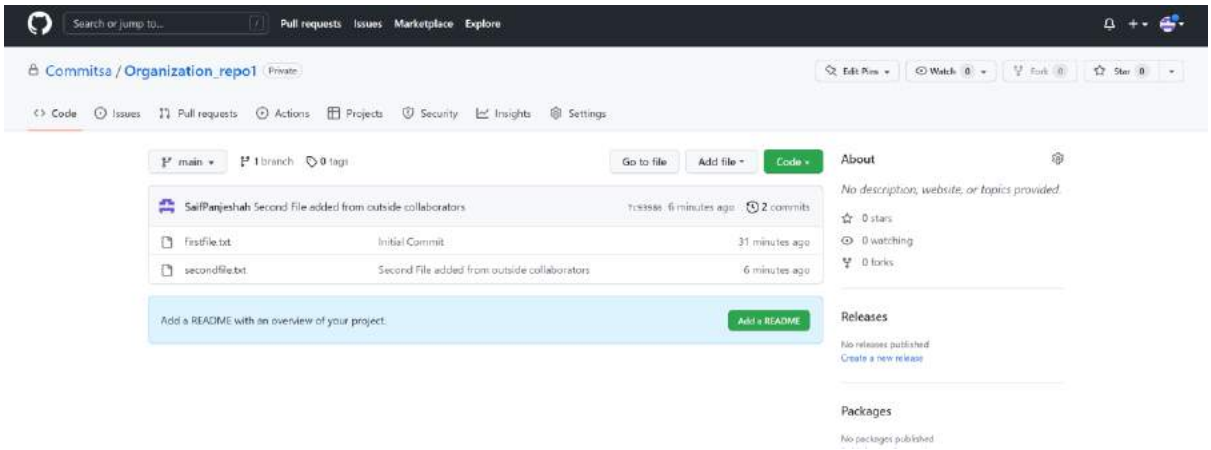


Fig. Successful added to GitHub

Adding Organization Members:

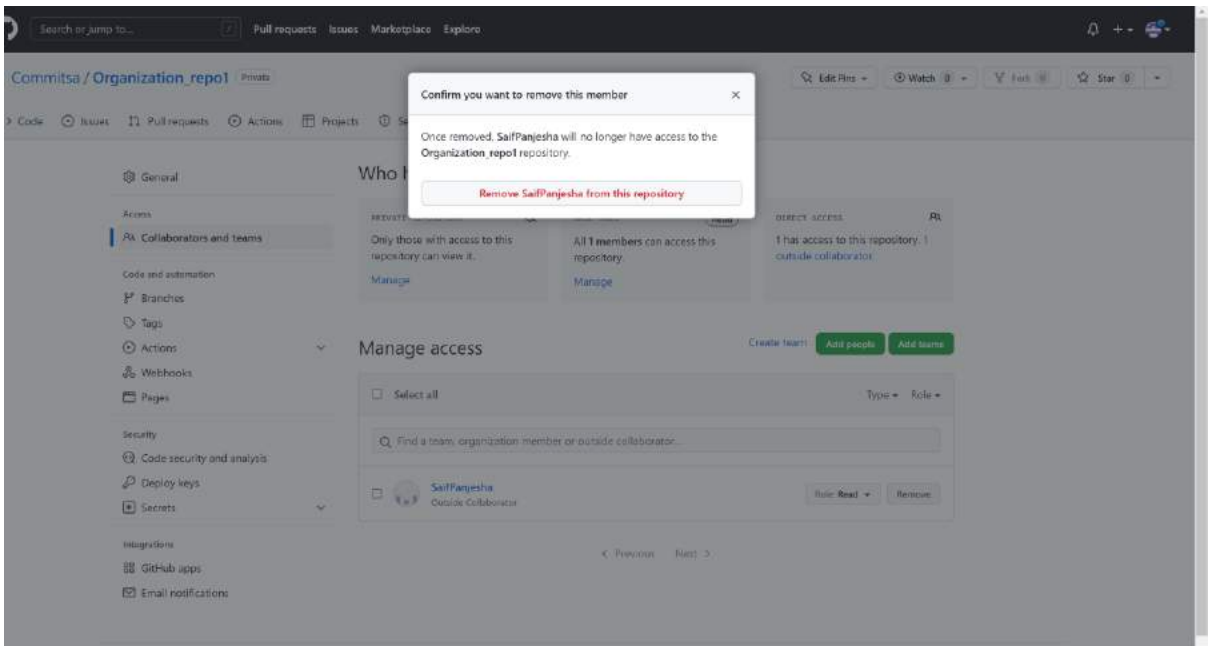


Fig. removing the outside collaborators

Let's explore people tab on Organization:

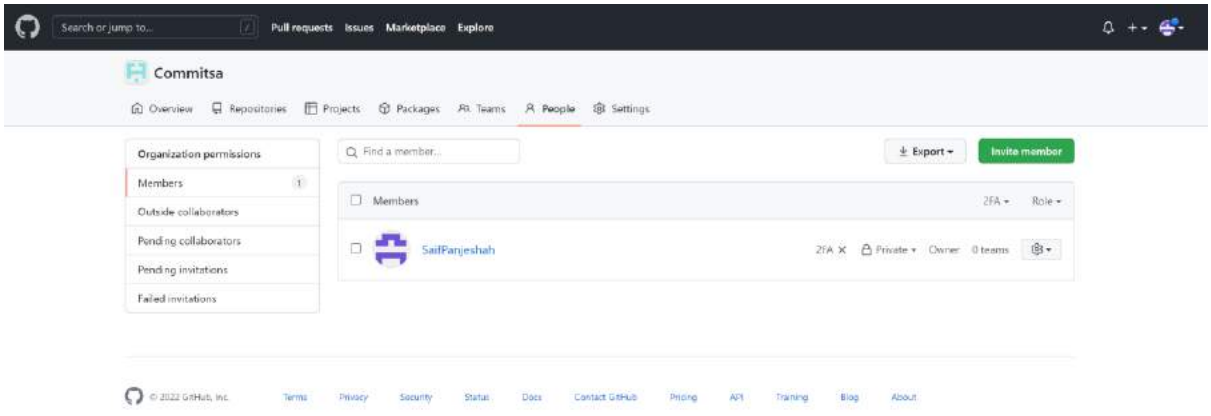


Fig. Explore people tab

Inviting Member:

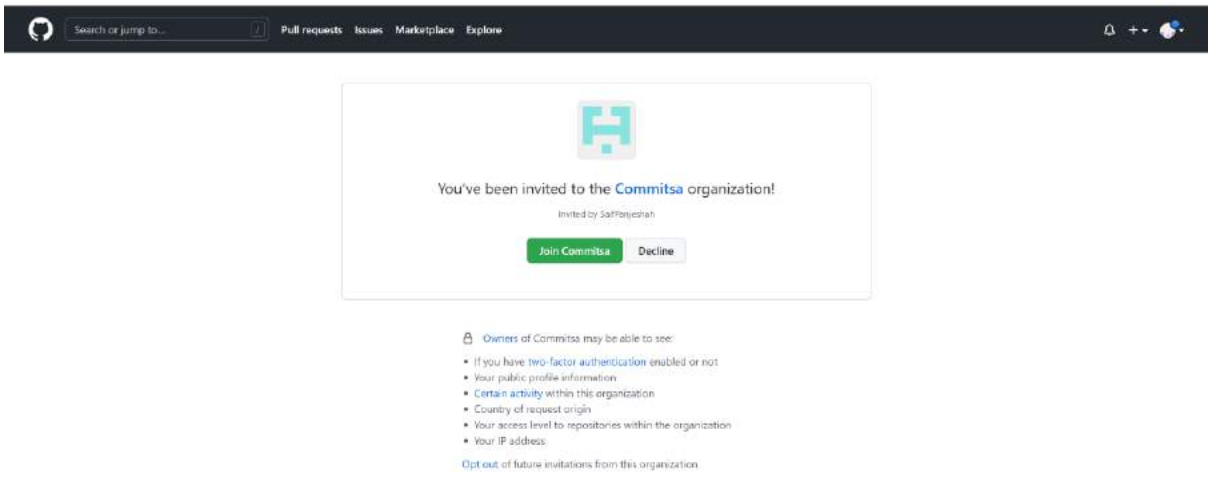


Fig. Inviting member to Commitsa organization

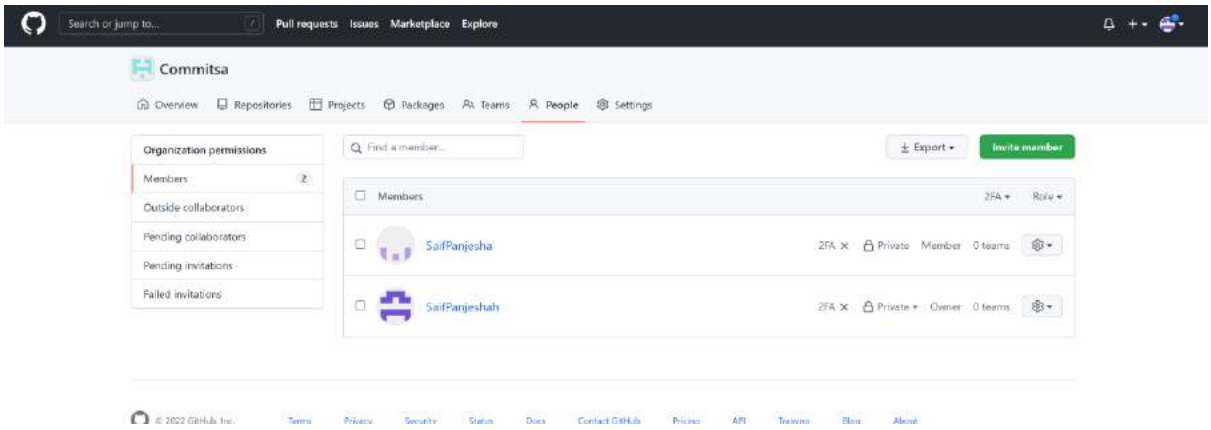


Fig. Successful Invitation

Now, trying to push the code from local branch git (VS Code) to check whether the code is successful to cloud or not.

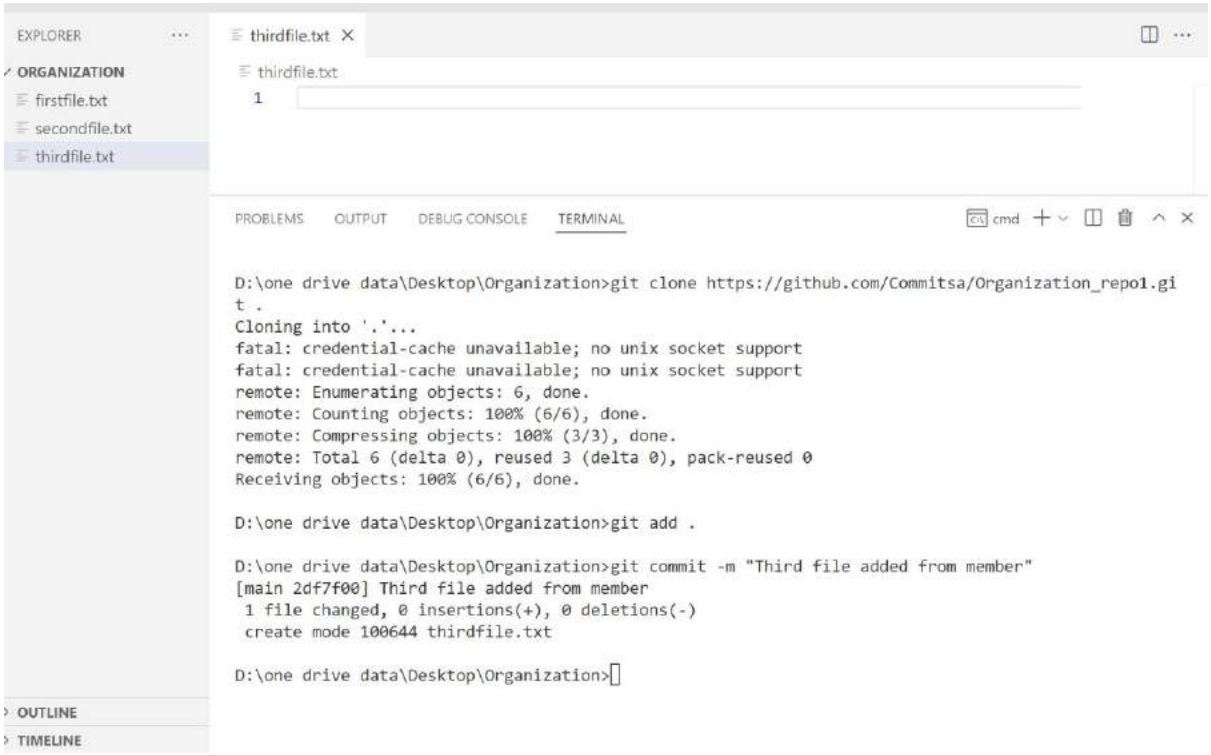


Fig. Successful clone

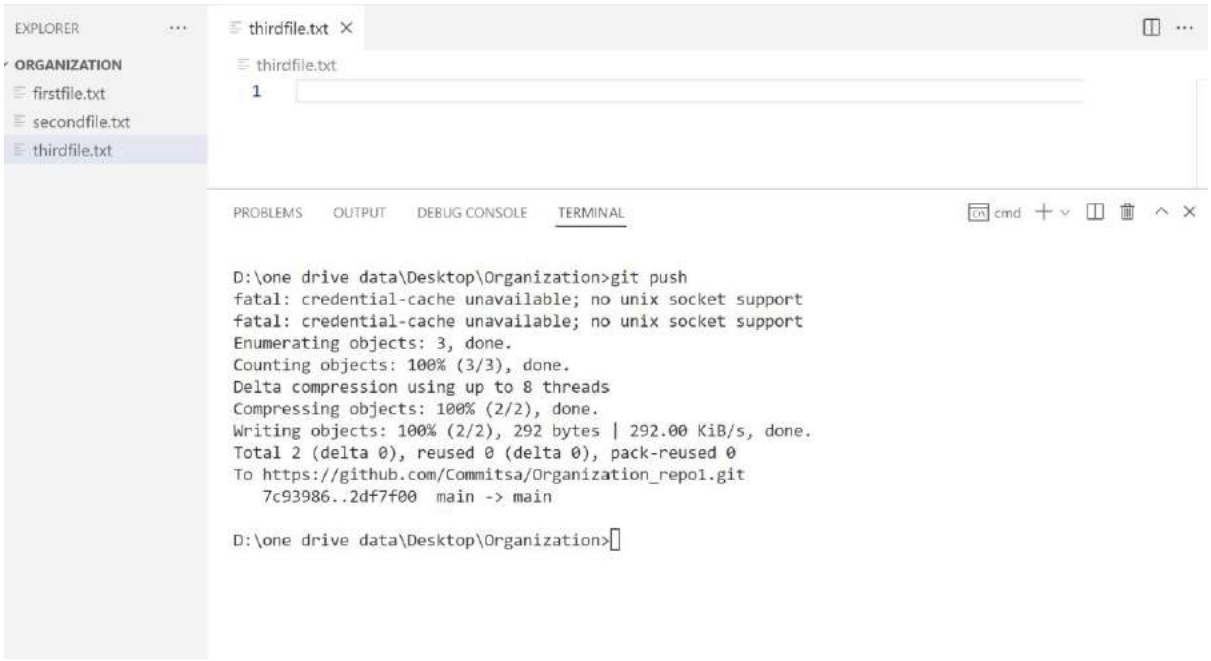


Fig. Successful Push

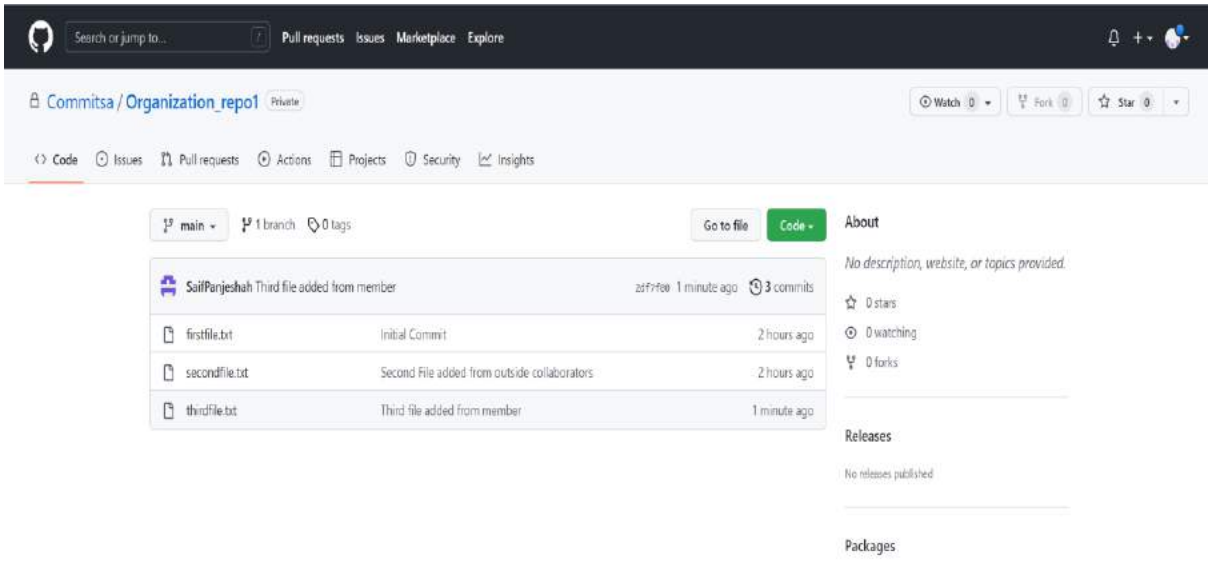


Fig. Successful Push on GitHub

Failing to Manage Access for Individual Repositories:

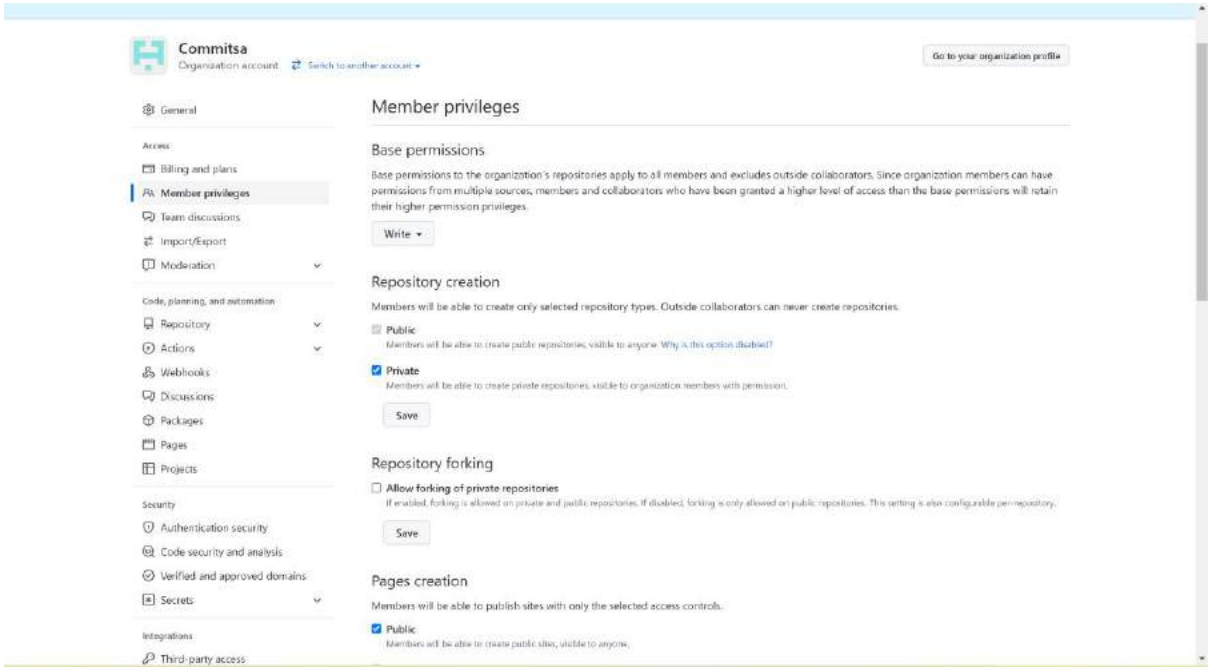


Fig. Giving Member privileges access to Write Option

Creating New Repositories in Organizational Account for read only members:

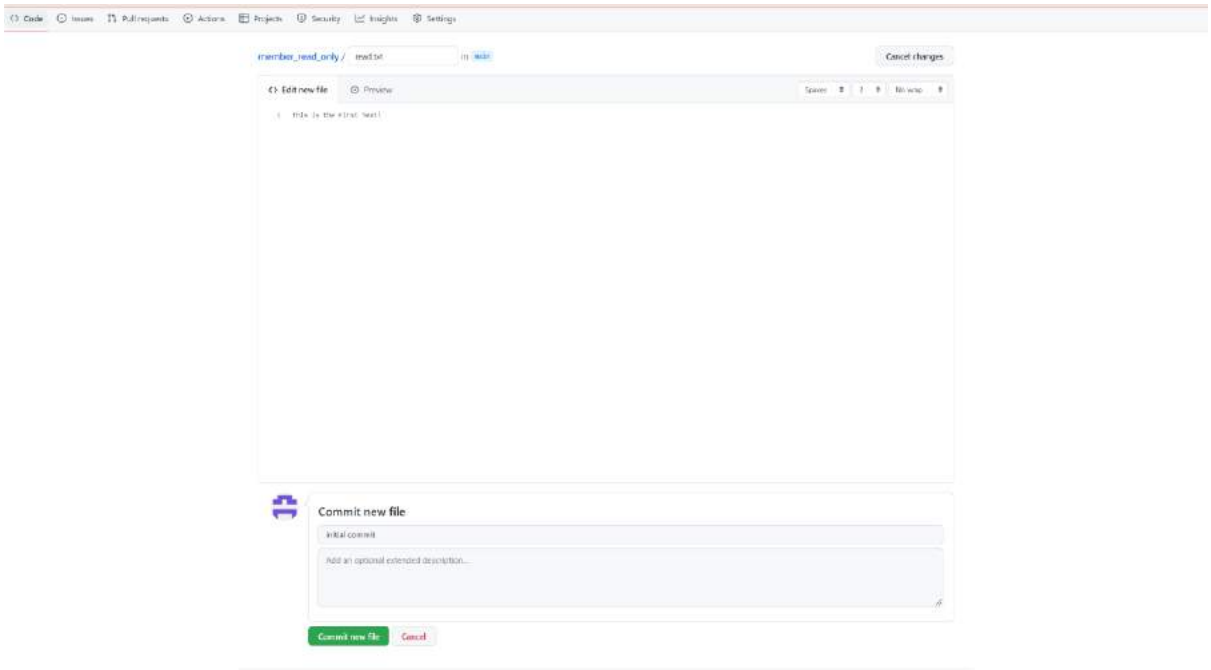


Fig. Creating member read only repositories

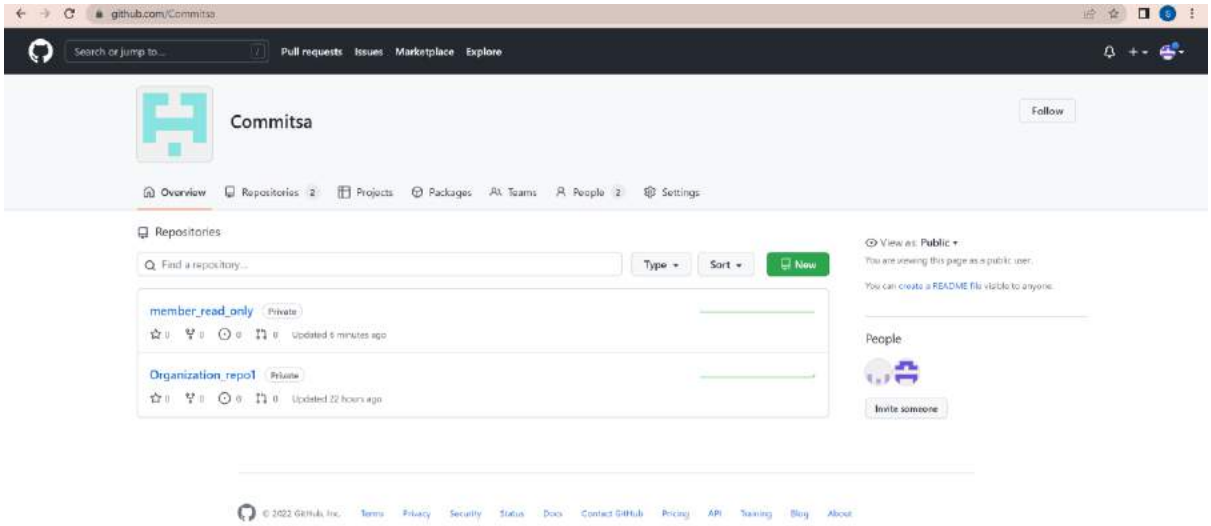


Fig. Successful Created Account

Checking Access for Individual Repositories:

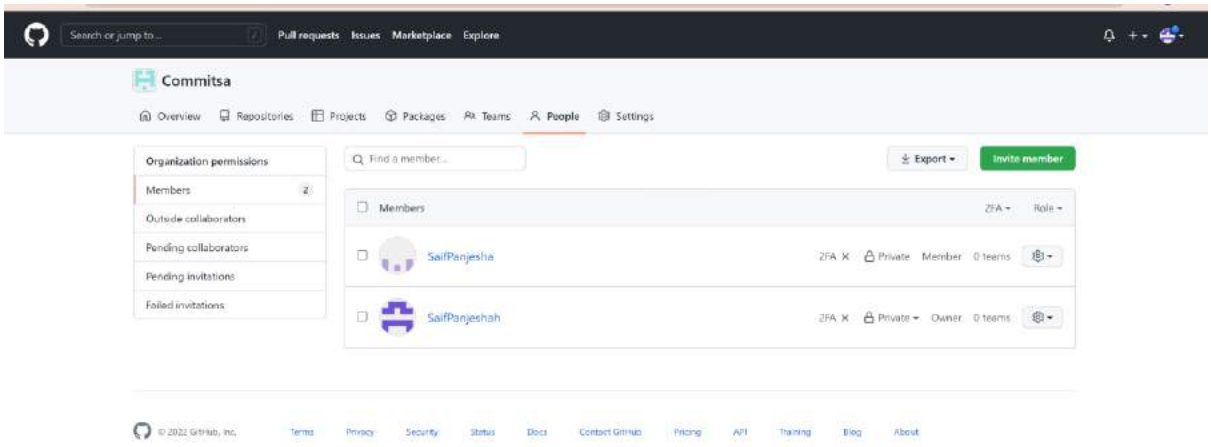


Fig. Checking access

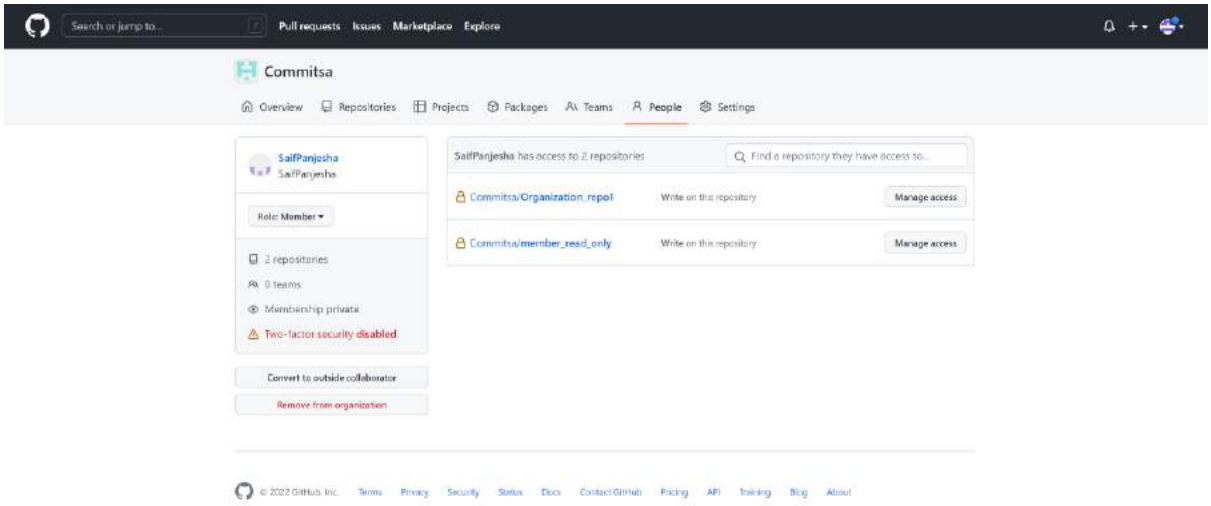


Fig. Shows both the account has same access for repositories

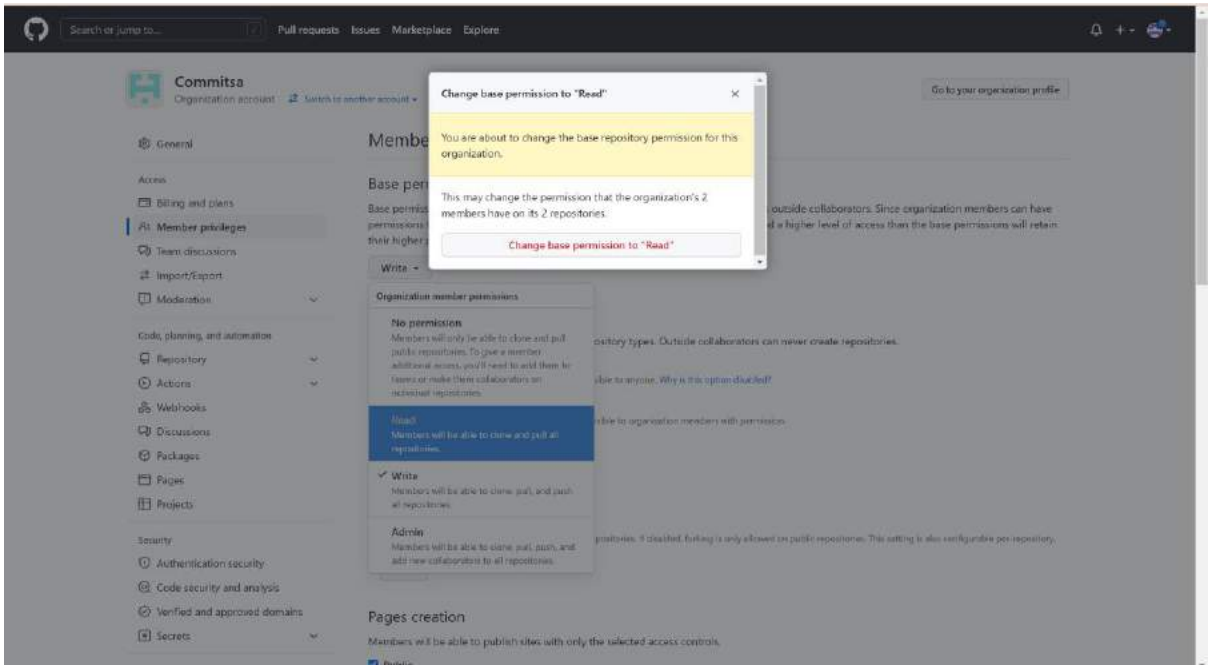


Fig.Shows Permission may change for both the repositories

How to resolve this problem access Permission?

By Introducing Teams in Organizational Account:

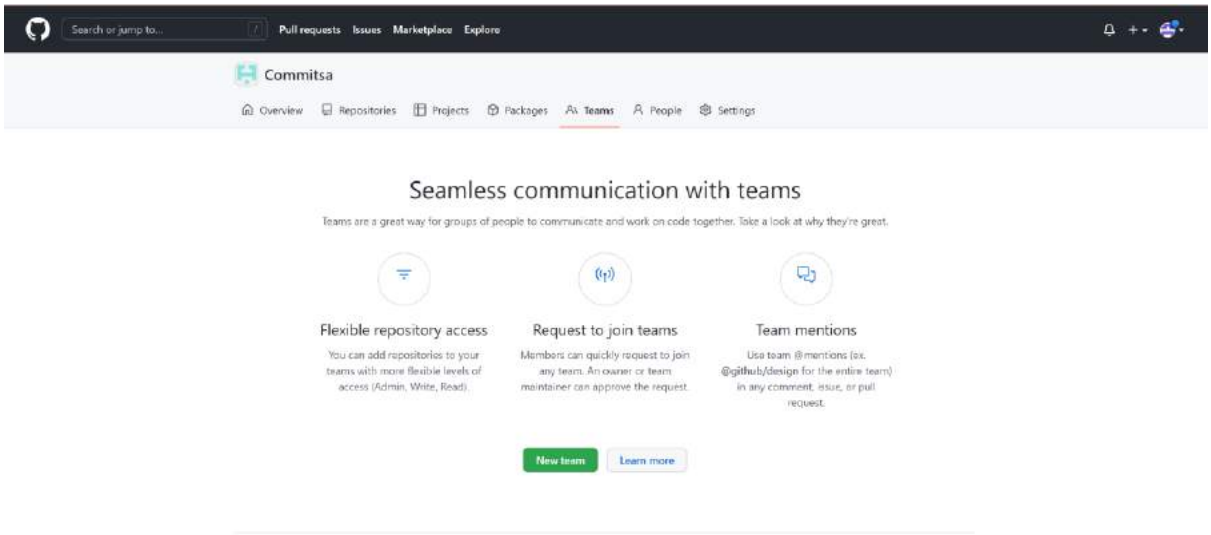


Fig. Teams Ribbons of Organizational Account

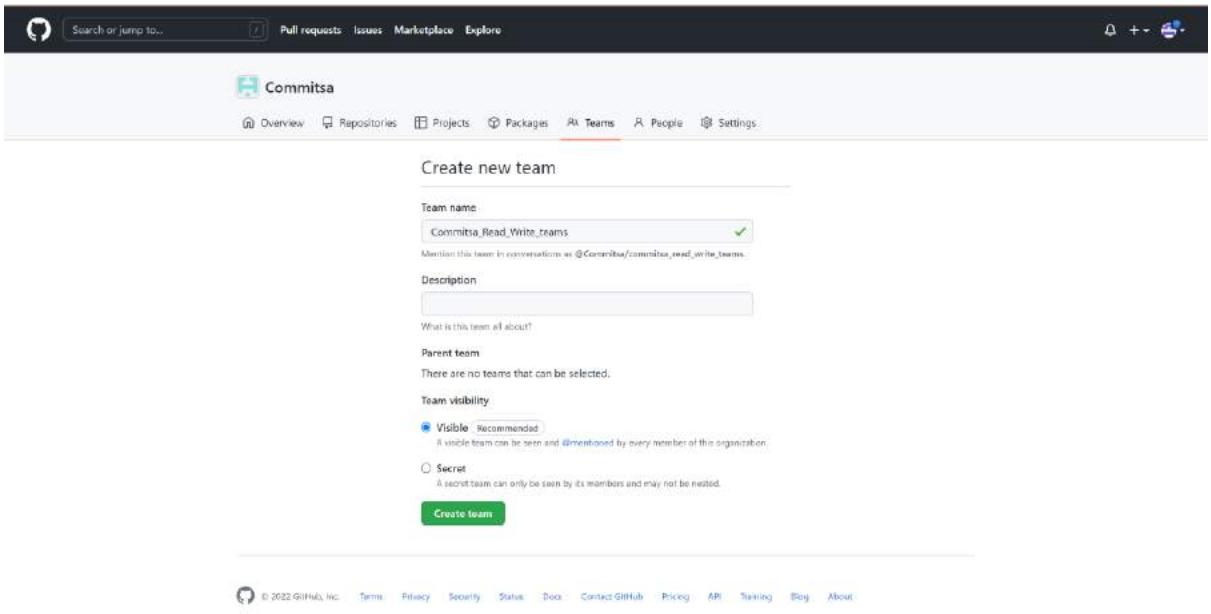


Fig. Creating Teams in Organization

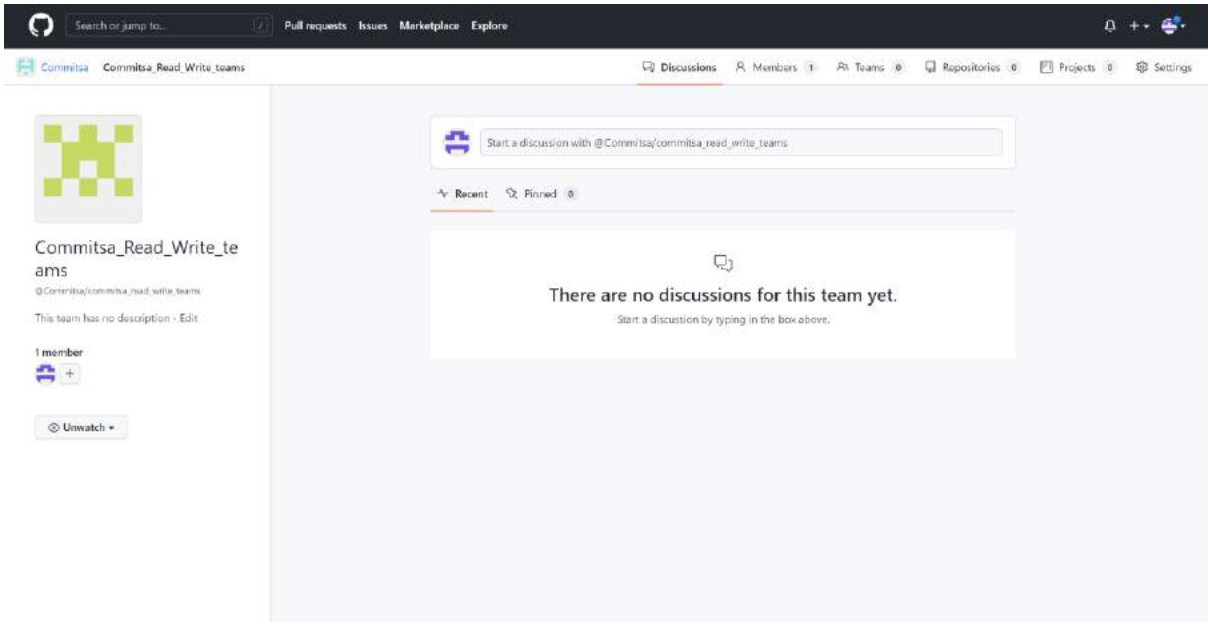


Fig. Read and Write team successfully Created

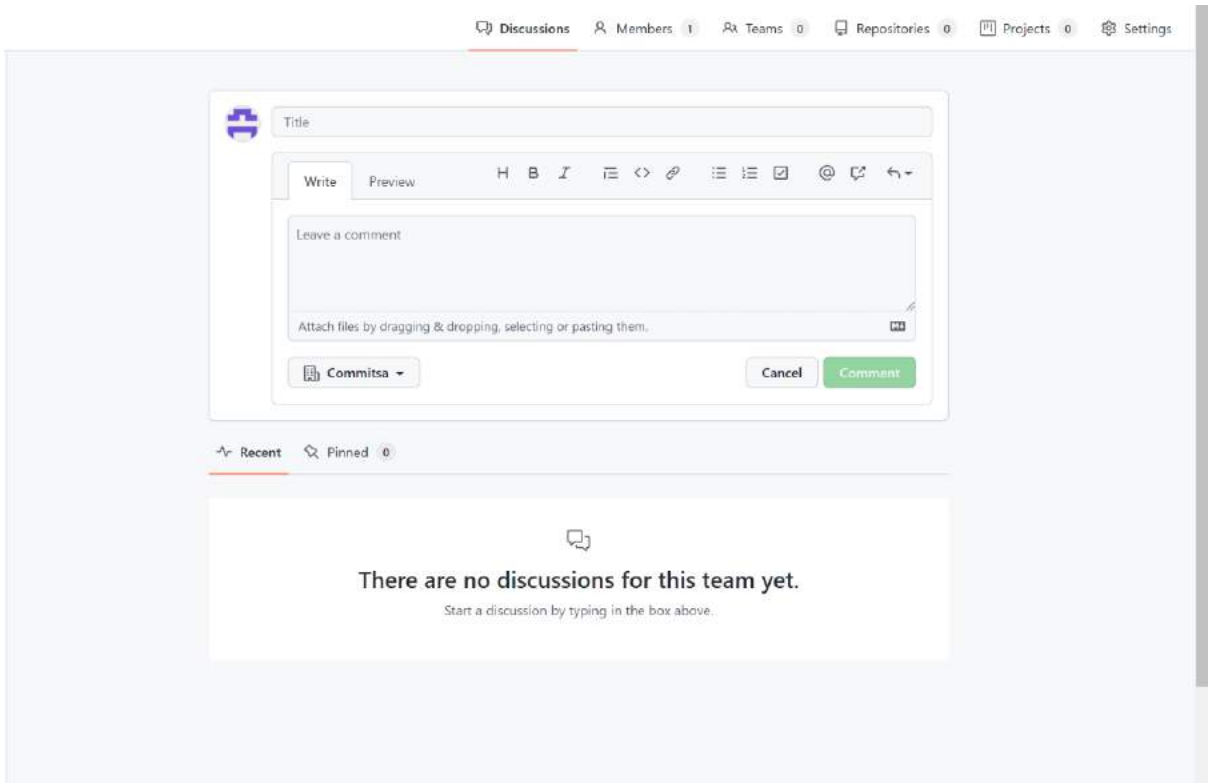


Fig. Forum of discussion ribbons in teams to check

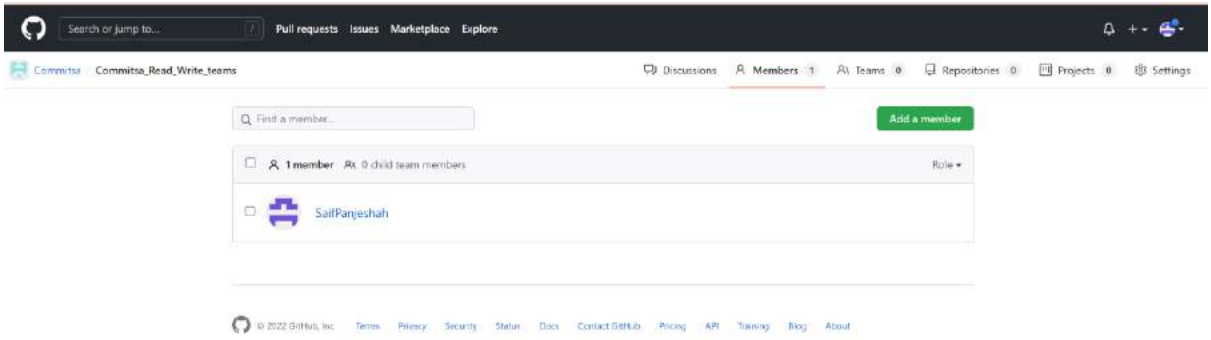


Fig. Owner as Team Member Default

Let's create team repository:

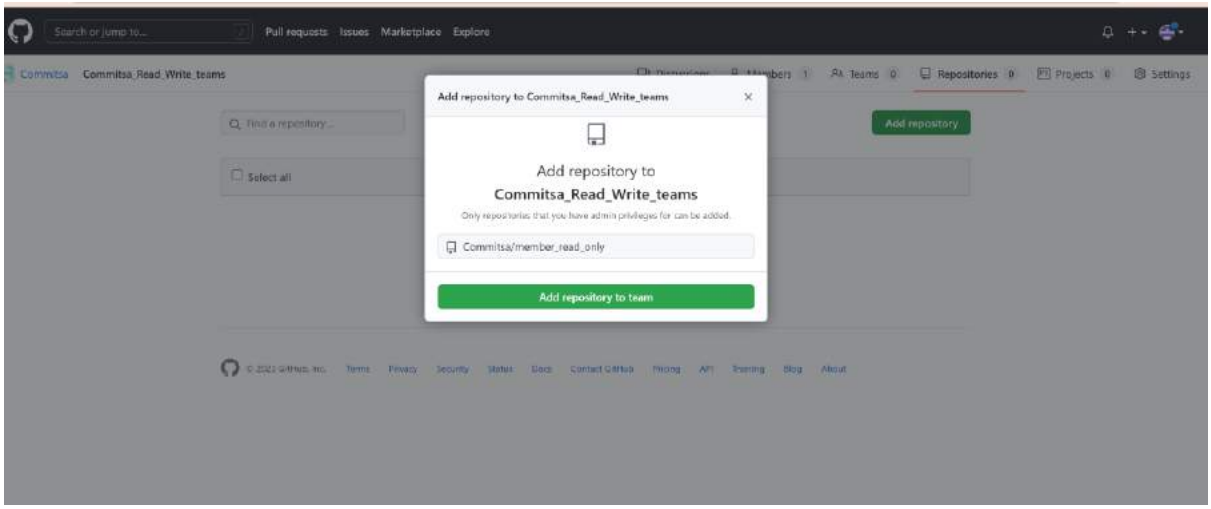


Fig. Creating Team Repository

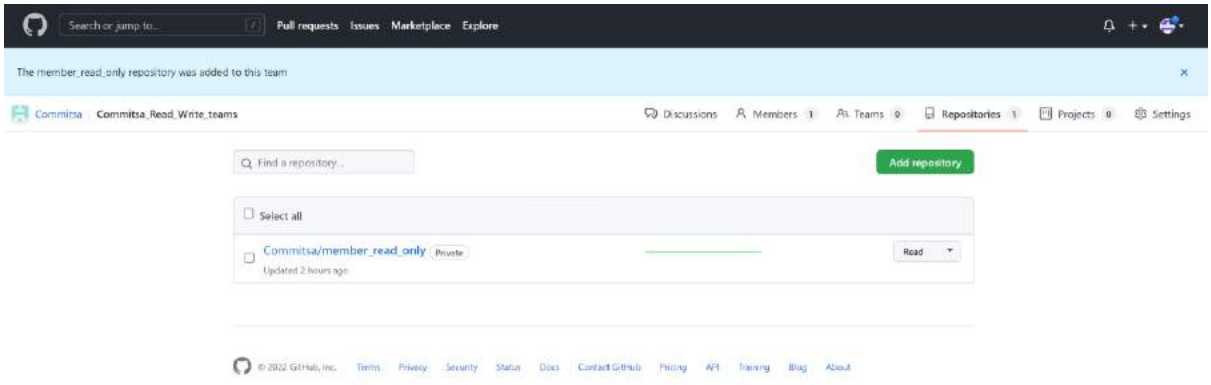


Fig. Repository Created with read access

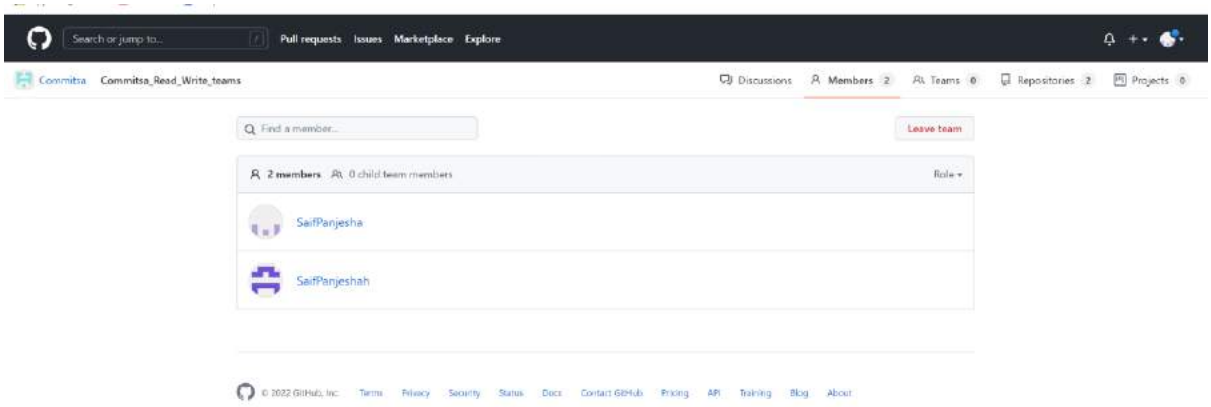


Fig. Successful adding team member

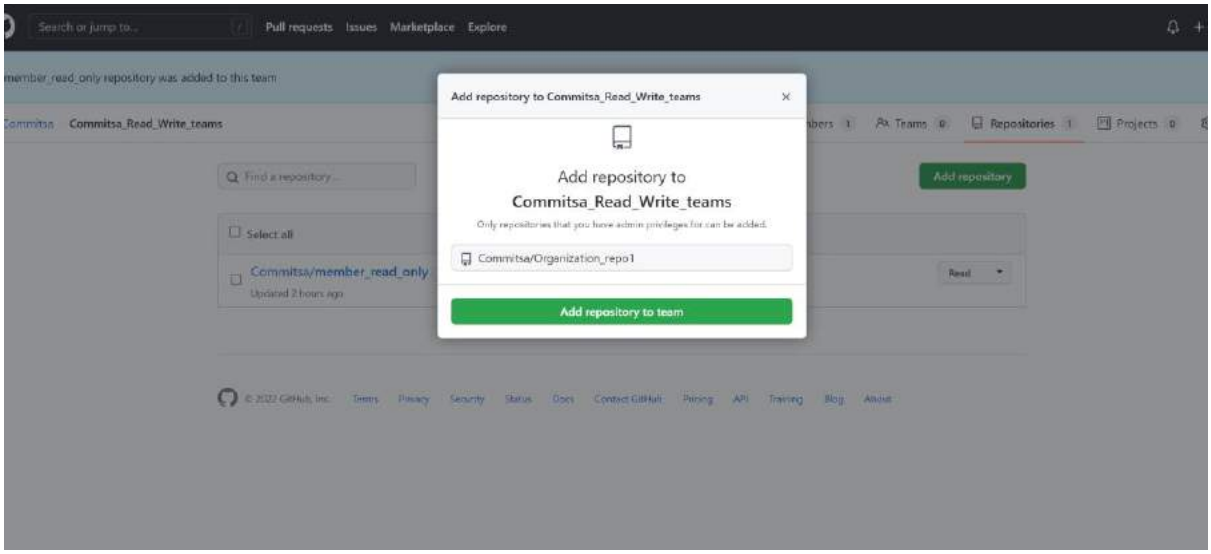


Fig. Creating Team Repository

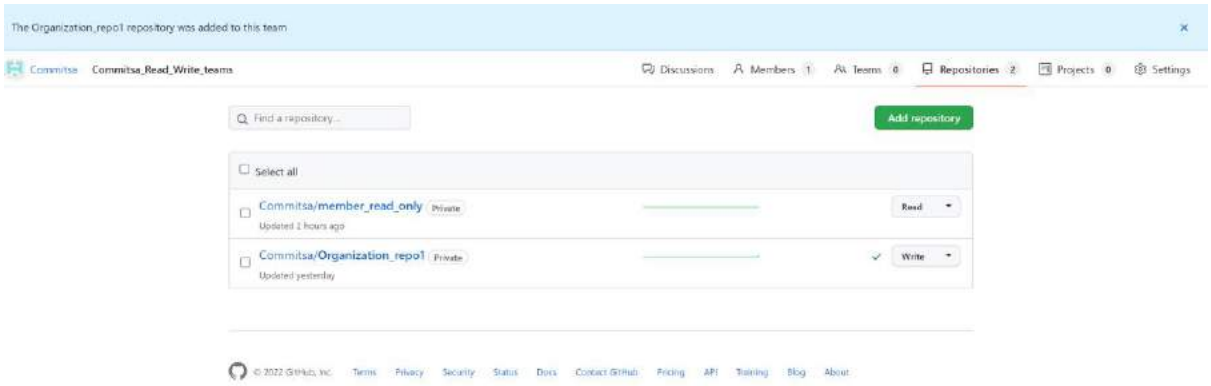


Fig. Successful Created repositories with write access

Hence, we can able to access for Individual Repositories with teams as flexible work.

Managing Team Repositories Access Efficiently:

Let's Now, trying to push the code from local branch git (VS Code) to check whether the code is successful to cloud or not.

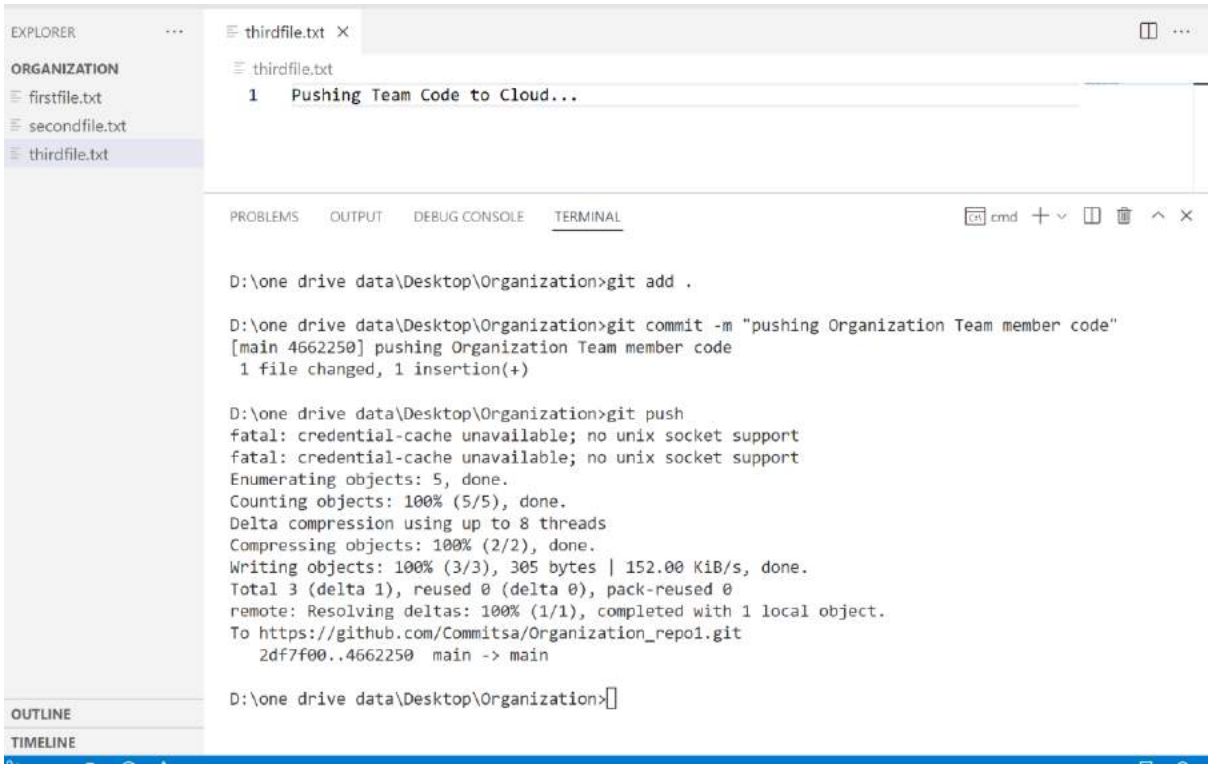


Fig. Successful Push coding on Organization repository with write access

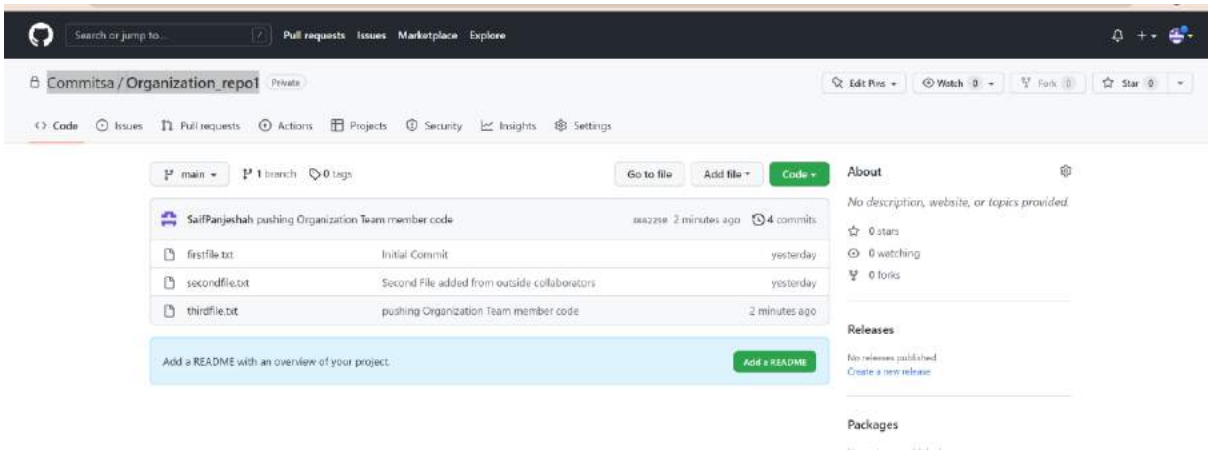


Fig. Successful Push on GitHub

Let's Push our code on: https://github.com/Commitsa/member_read_only.git.

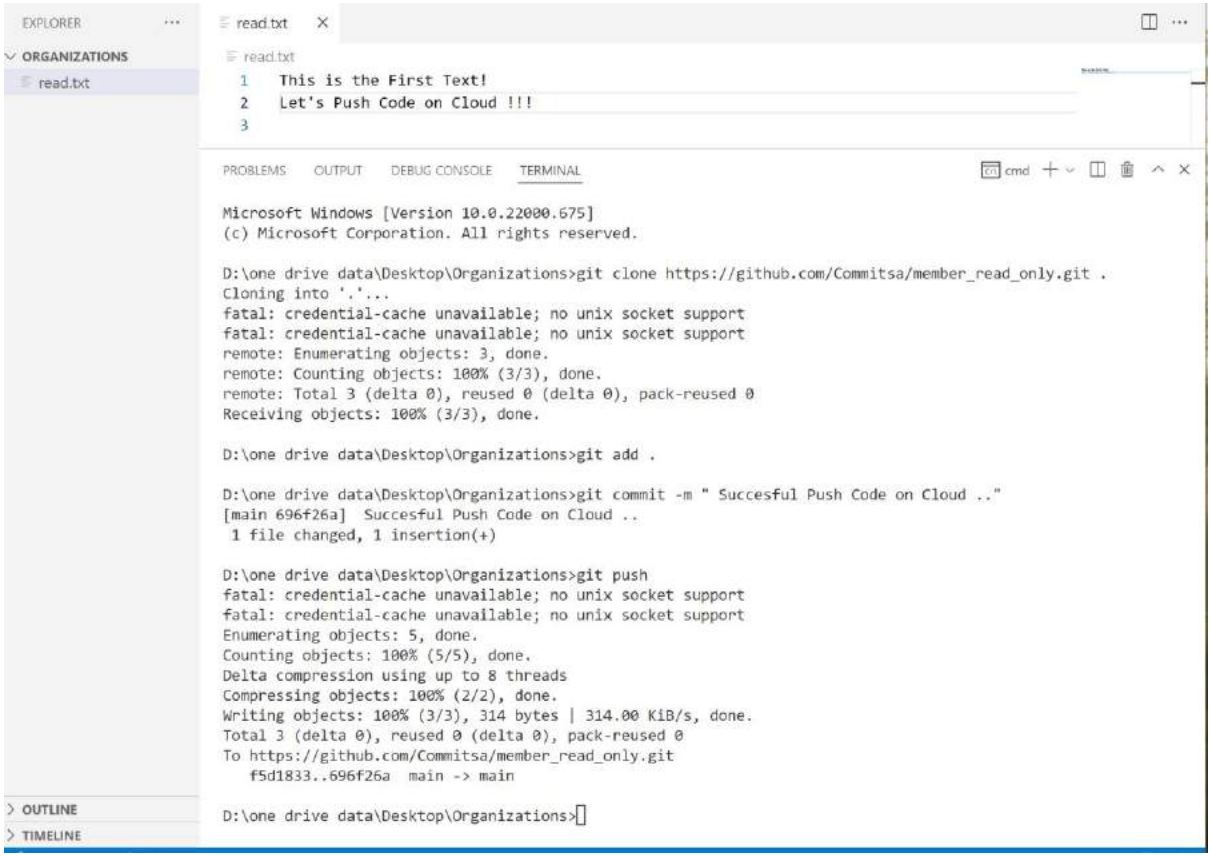


Fig. Successful Push coding on member_read_only repository with read access

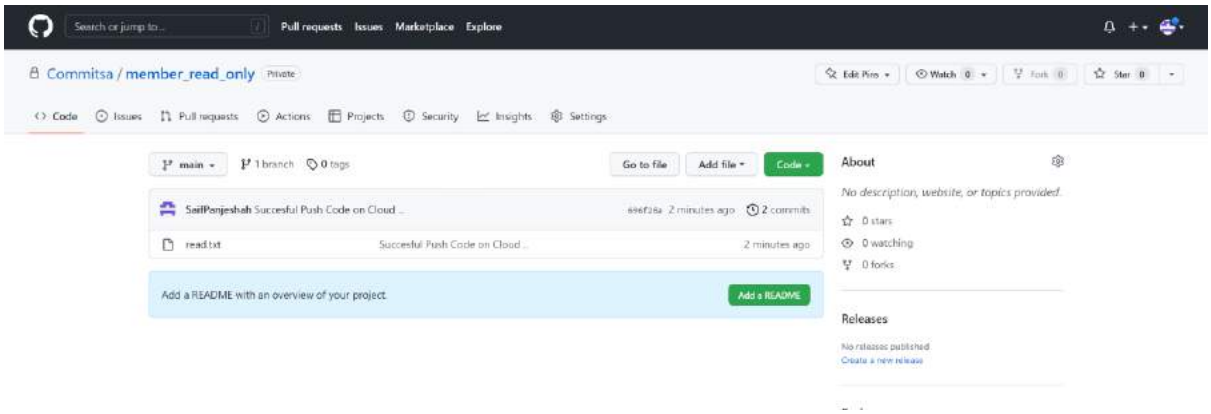


Fig. Successful Push on GitHub

Understanding Forks and Pull Request:

Definition: A fork is a copy of a repository that you manage. Forks let you make changes to a project without affecting the original repository. You can fetch updates from or submit changes to the original repository with pull requests.

Fork	Clone
Forking is done on the GitHub Account	Cloning is done using Git
Forking a repository creates a copy of the original repository on our GitHub account	Cloning a repository creates a copy of the original repository on our local machine
Changes made to the forked repository can be merged with the original repository via a pull request	Changes made to the cloned repository cannot be merged with the original repository unless you are the collaborator or the owner of the repository
Forking is a concept	Cloning is a process
Forking is just containing a separate copy of the repository and there is no command involved	Cloning is done through the command '**git clone**' and it is a process of receiving all the code files to the local machine

Fig. Difference between Fork & Clone

When to use: A fork is a rough copy of a repository. Forking a repository allows you to freely test and debug with changes without affecting the original project. One of the excessive uses of forking is to propose changes for bug fixing. To resolve an issue for a bug that you found, you can:

- Fork the repository.
- Make the fix.
- Forward a pull request to the project owner

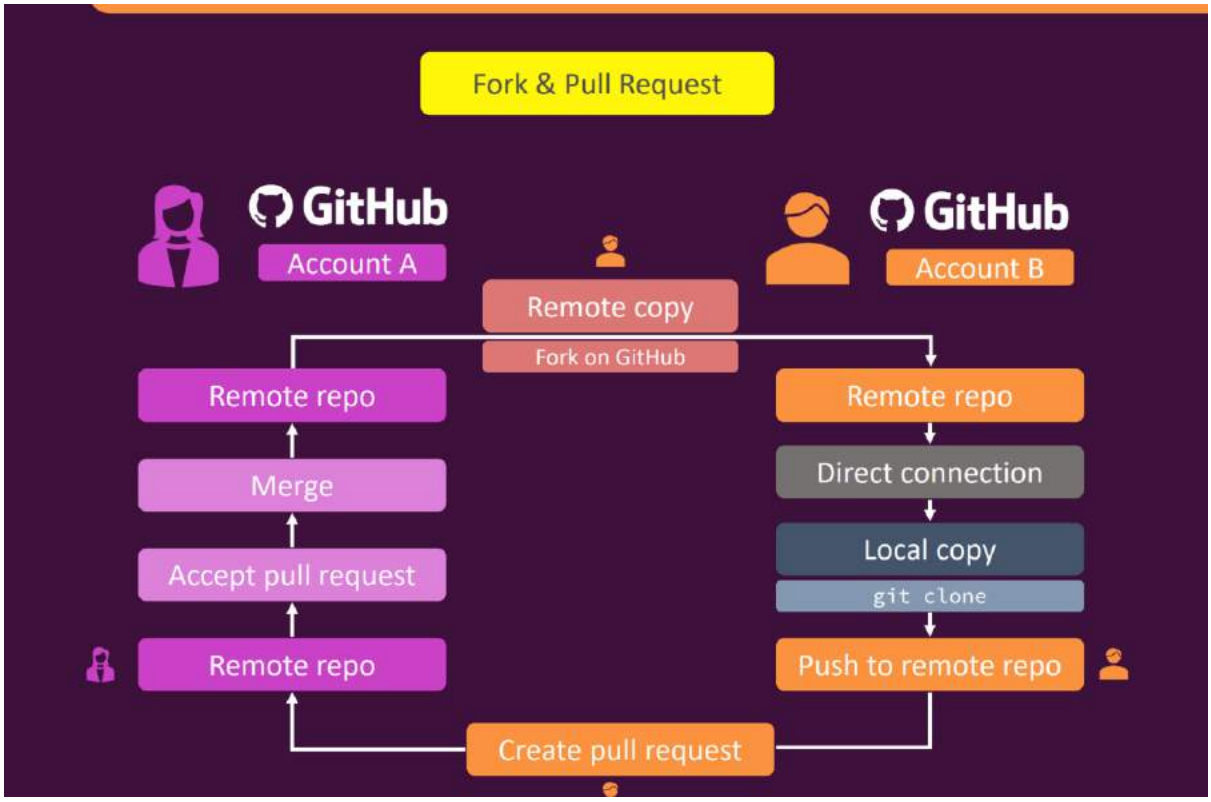


Fig. Creating Fork & Pull Request

1. Forking the Repository:

Assuming you're using GitHub, this step is easy. Just find the repository you're contributing to and press the Fork button in the upper right. This will create an exact copy of the repository (and all of its branches) under your own username

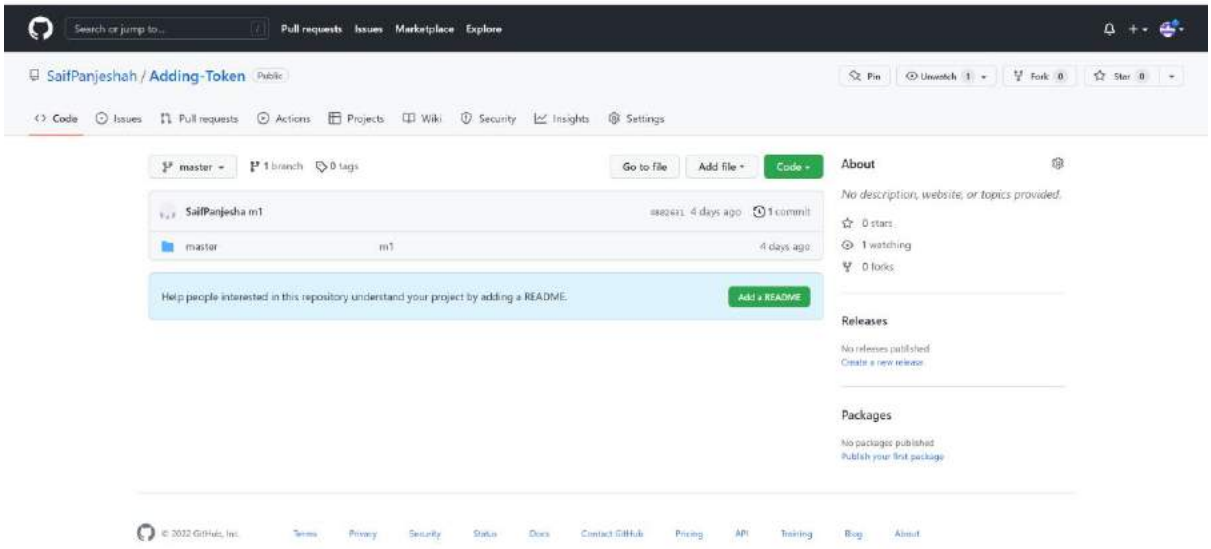


Fig. Forking the repository from another account

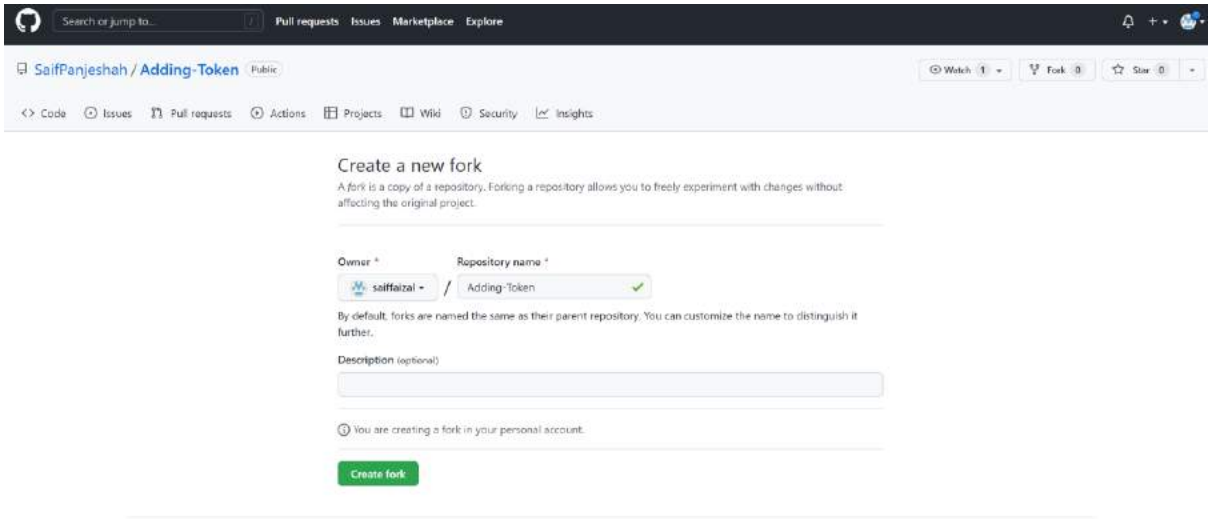


Fig. Create a new fork

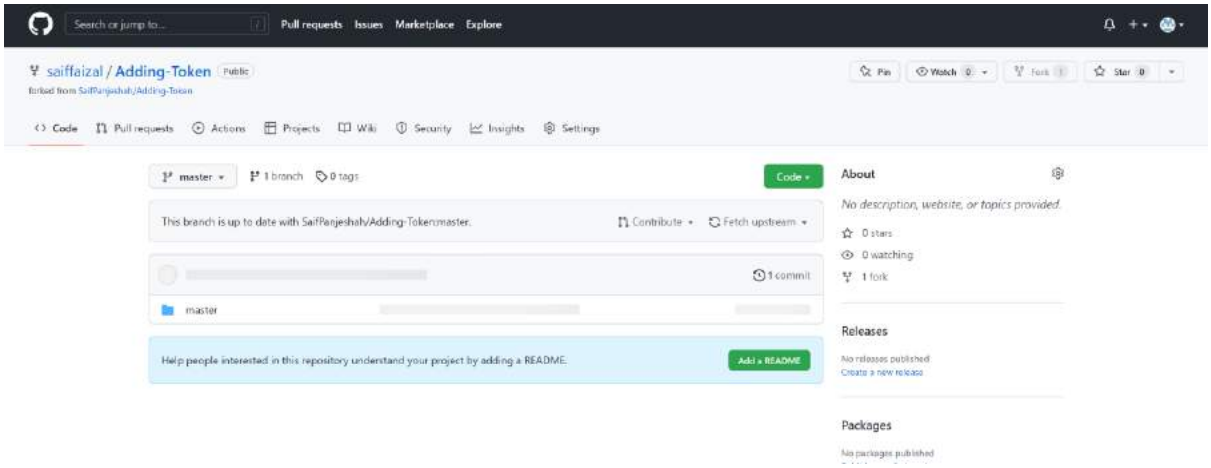


Fig. Forked Successful

2. Clone your new fork locally

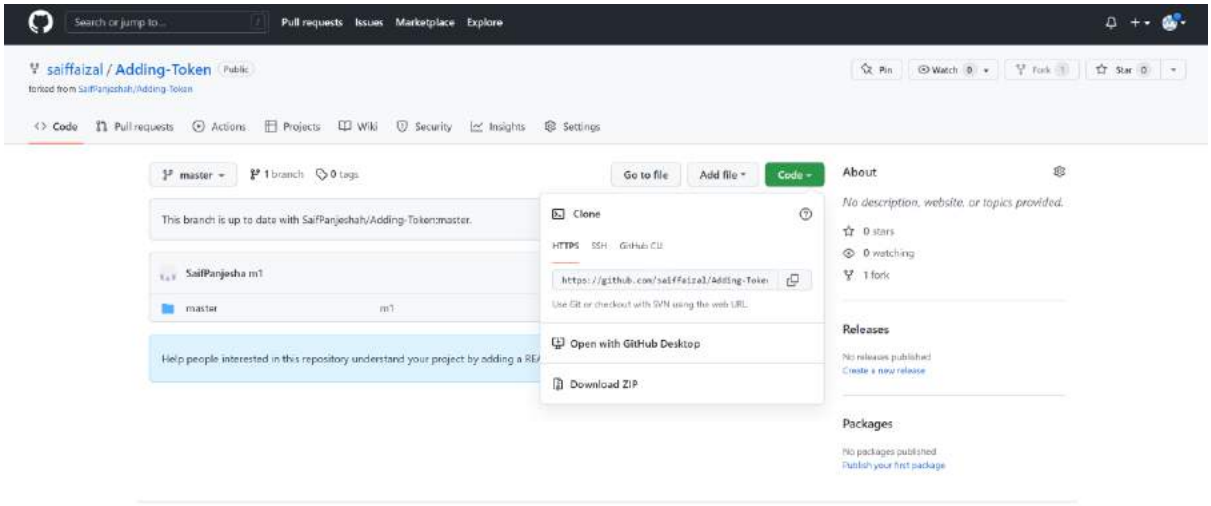


Fig. cloning the repository.

Let's Now, trying to push the code from local branch VS code console to check whether the code is successful to cloud or not.

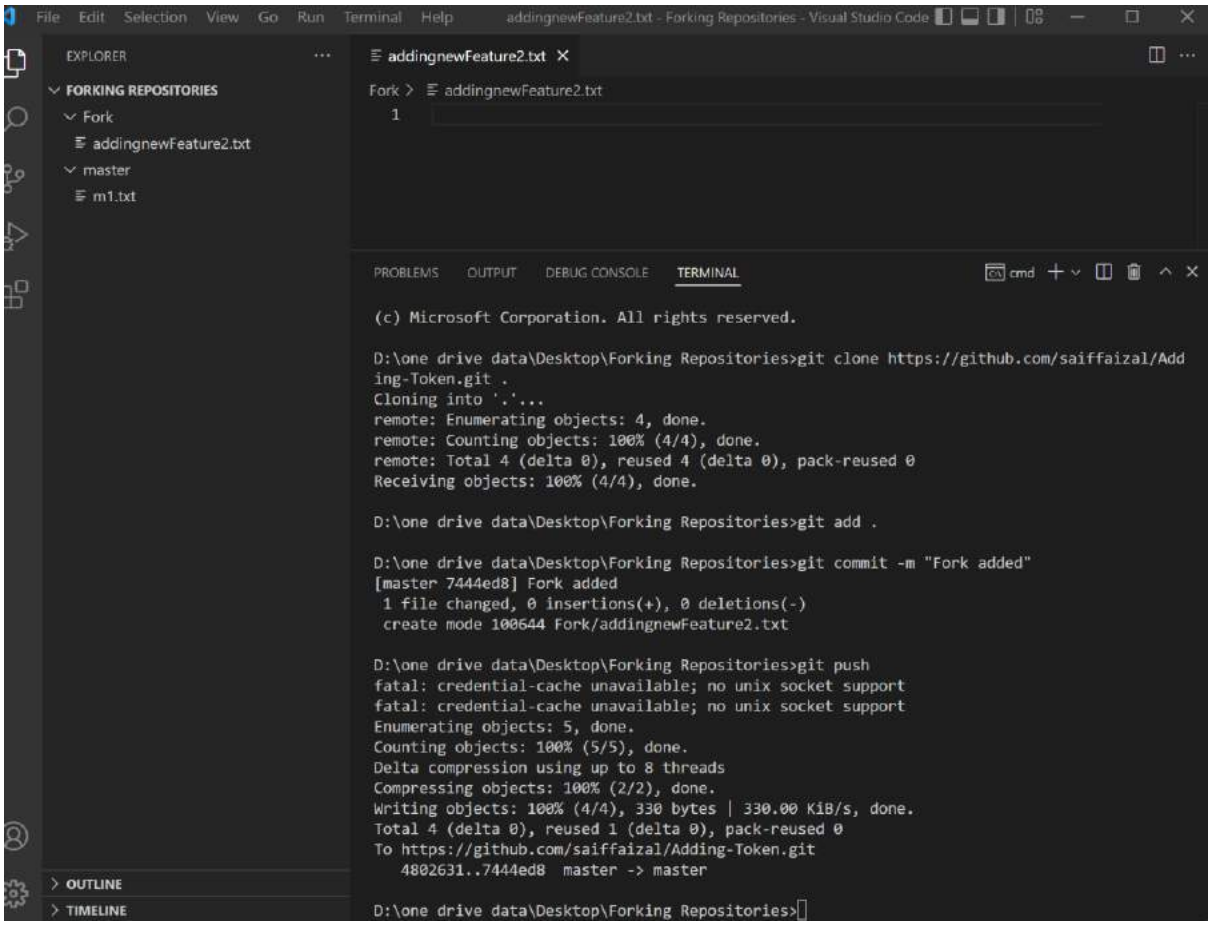


Fig. Success to push the code

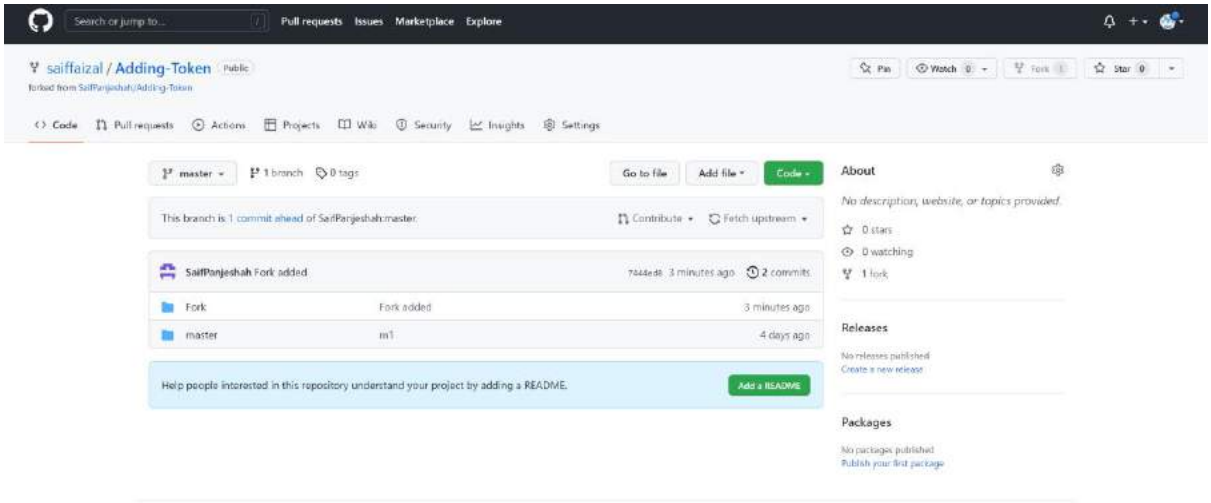


Fig. Successful on GitHub to member account

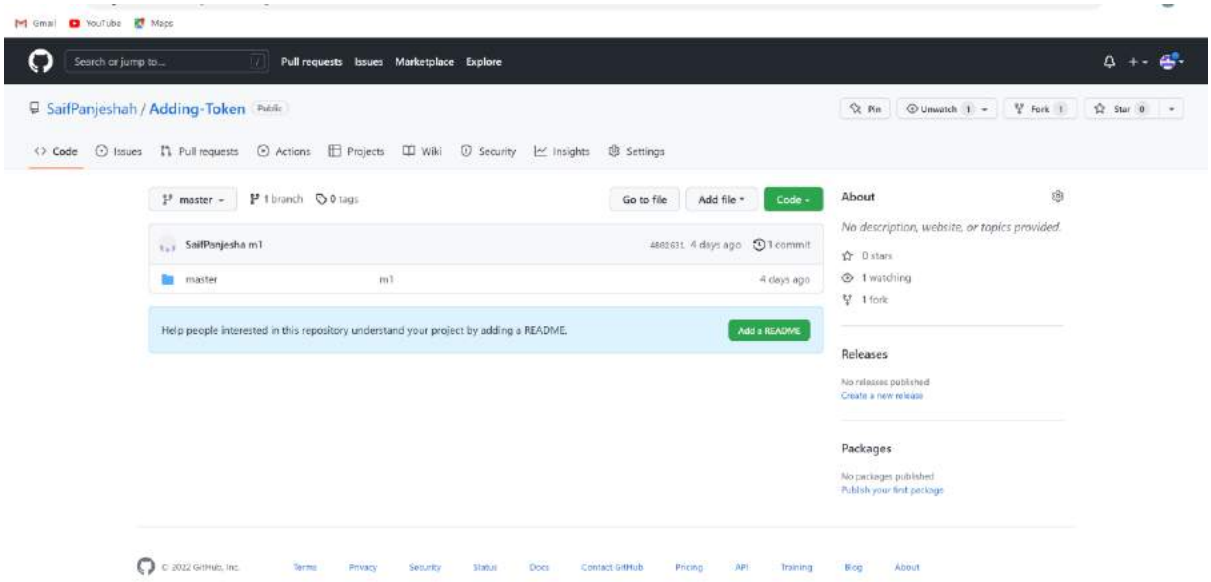


Fig. Not Fork Update Successful on GitHub owner account

How To resolve this problem?

By using pull requests in practise:

Creating a pull request

1. Switch to the branch that you want to create a pull request for Saiffaizal/Adding-Token

2. Click **Create Pull Request**. GitHub Desktop will open your default browser to take you to GitHub

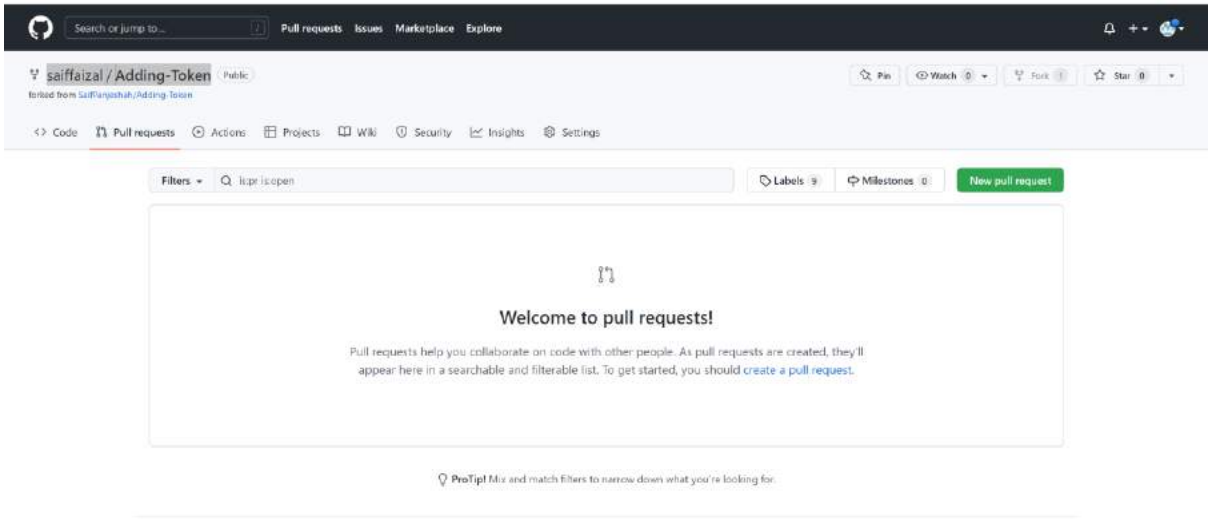


Fig. Creating a pull request

3. On GitHub, confirm that the branch in the **base:** drop-down menu is the branch where you want to merge your changes. Confirm that the branch in the **compare:** drop-down menu is the topic branch where you made your changes.

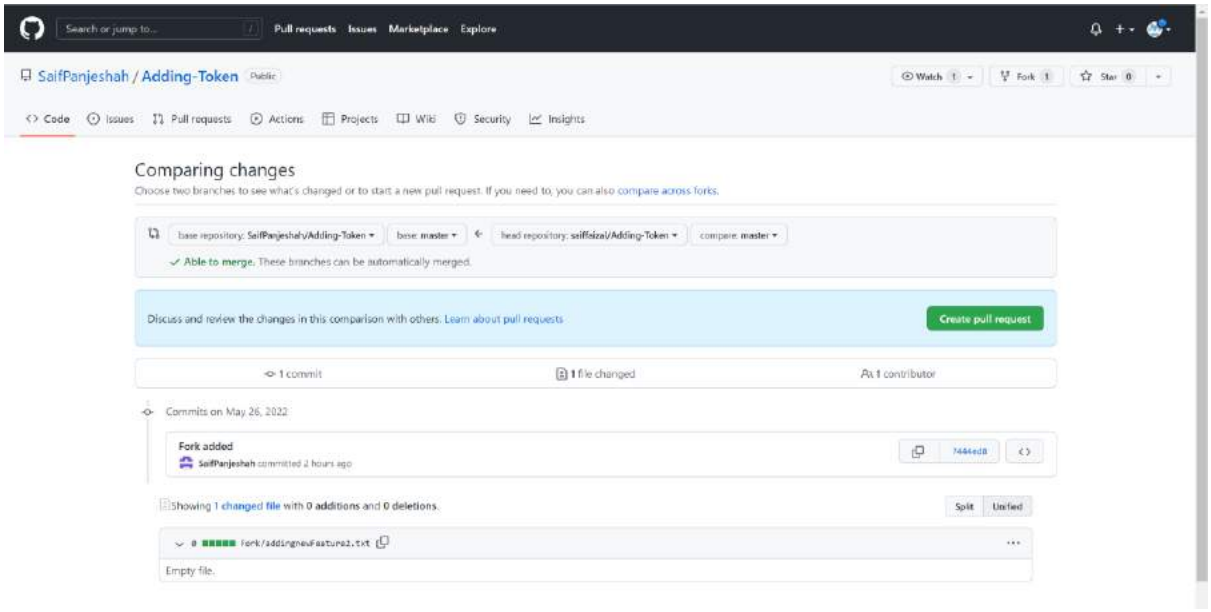


Fig. Comparing Changes

4. Type a title and description for your pull request.

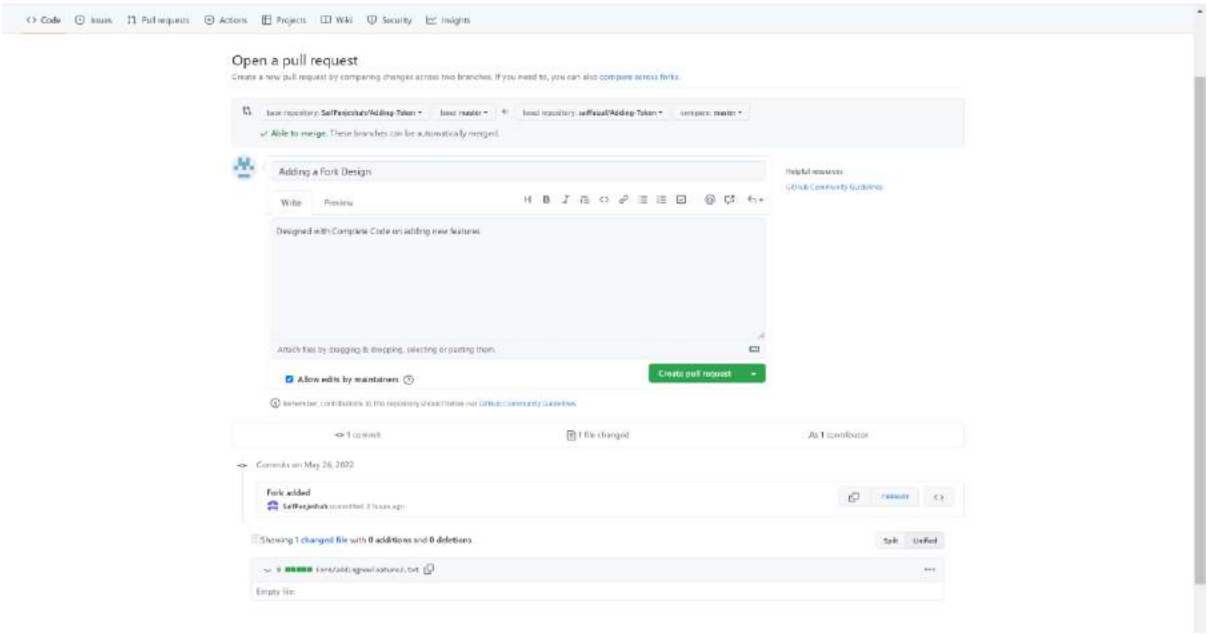


Fig. open pull request

5. **To create a pull request that is ready for review, click** Create Pull Request**.**

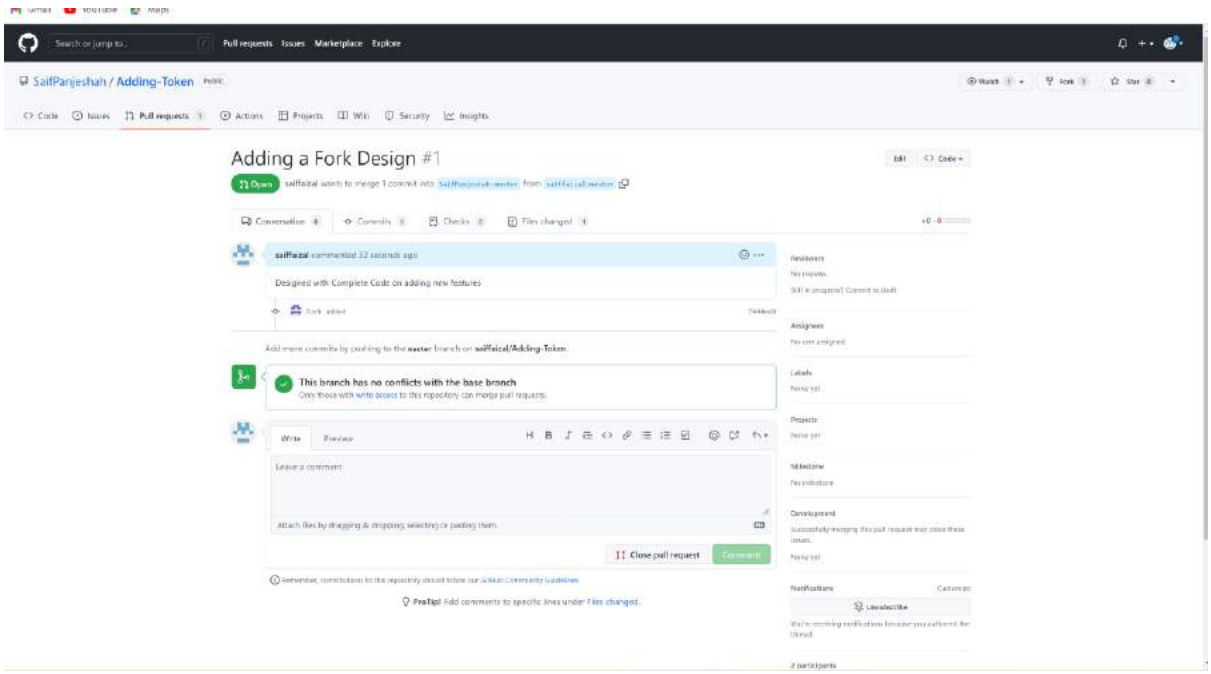


Fig. open pull request successful

Now, to view this pull request open owner organizational account

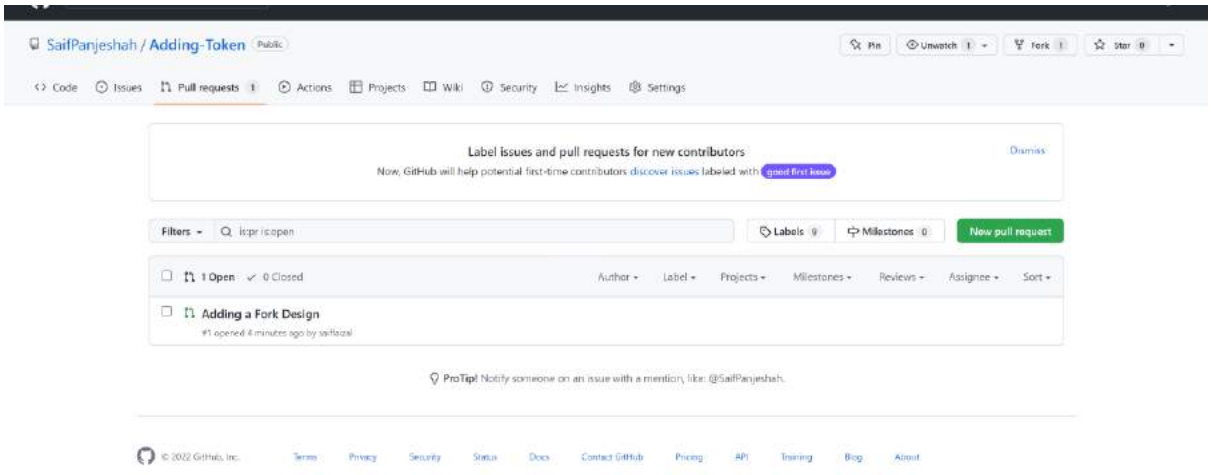


Fig. Owner view the pull request

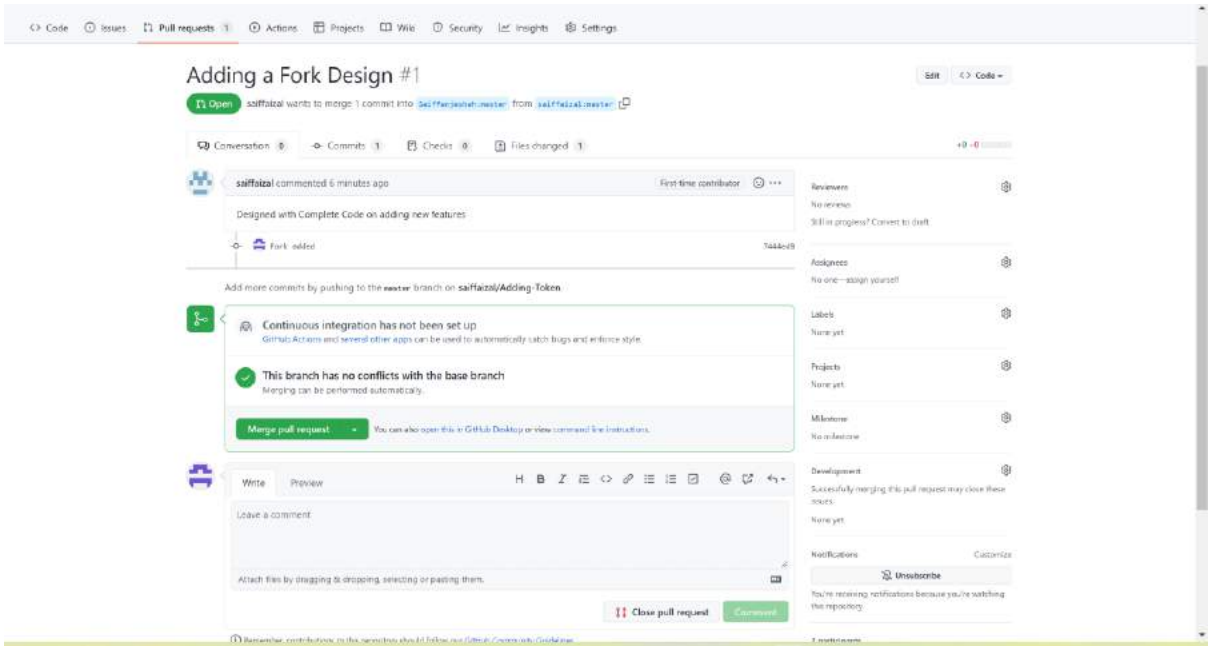


Fig. Merging or closing pull request

If owner don't like and don't want to merge this pull request simply close this pull request

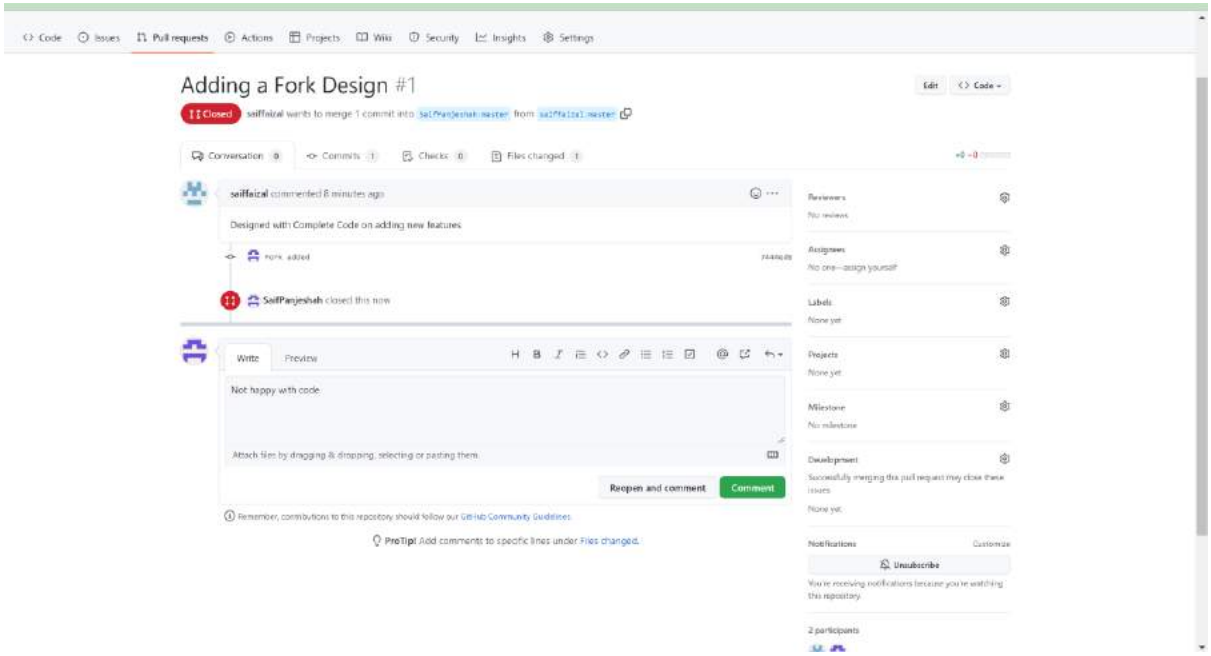


Fig. Owner closing this pull request

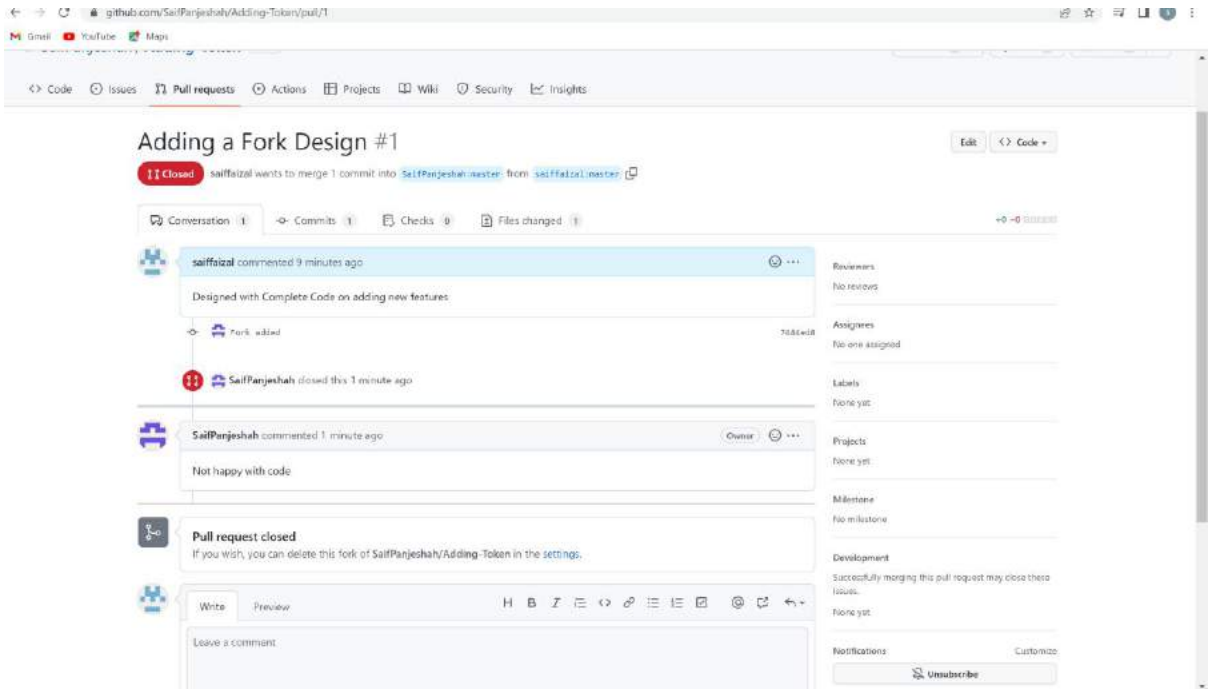


Fig. Closed Successful view from Member account

Now, Owner Thinks a Second Way like the Fork design and wants to merge the code

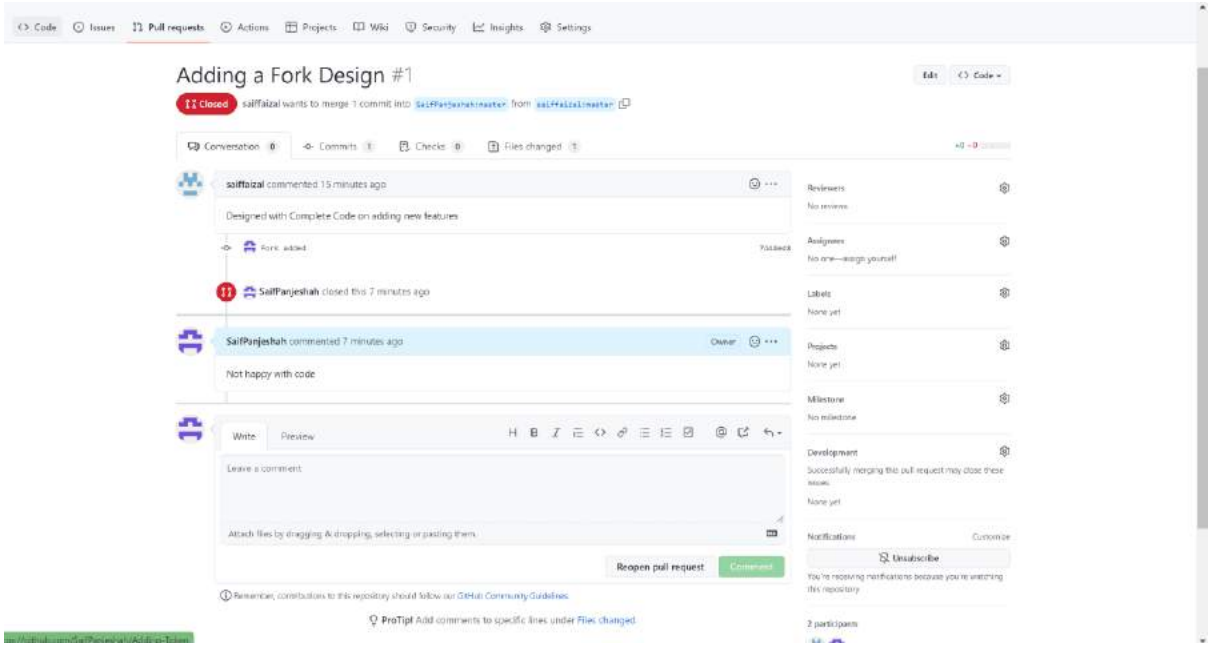


Fig. reopen the pull requests

Merging the pull requests:

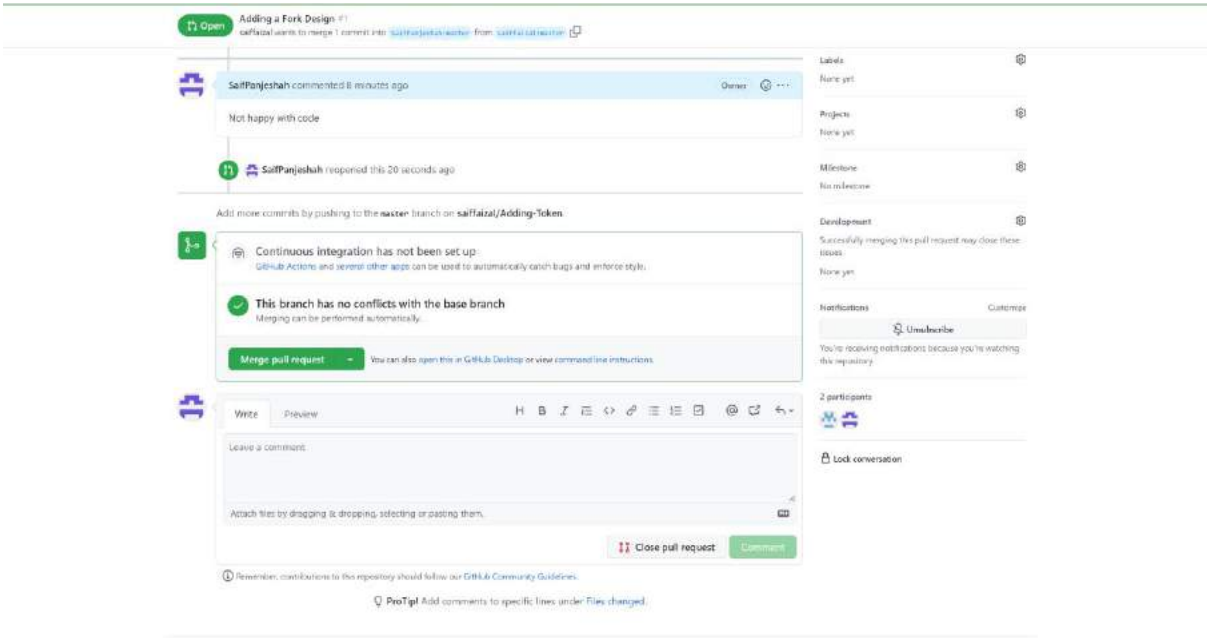


Fig. Merging the pull requests

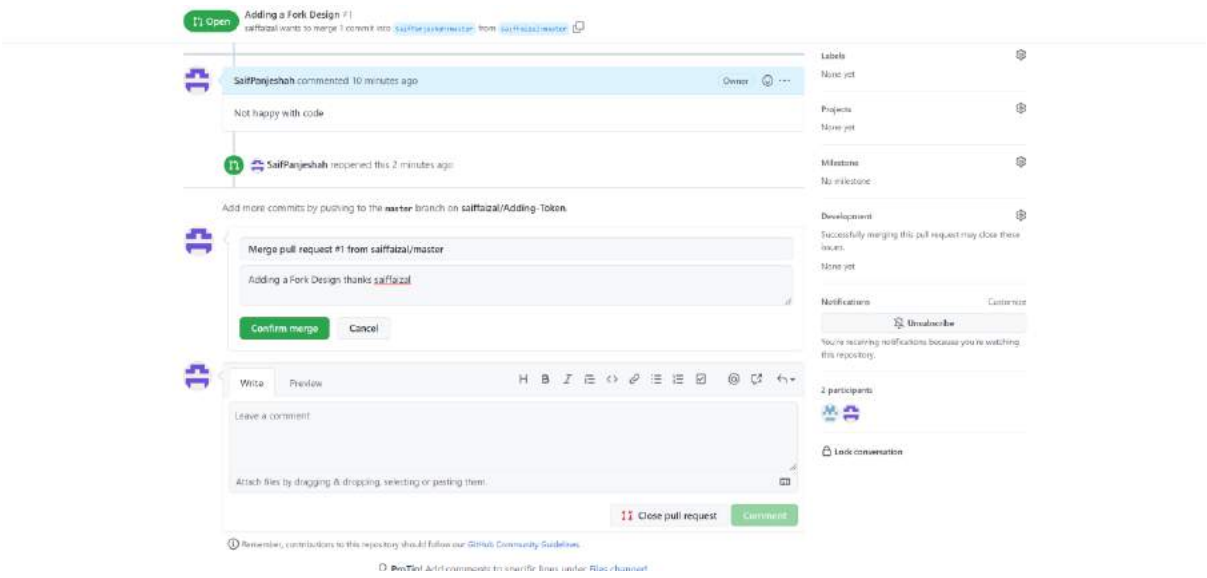


Fig. Confirm merging

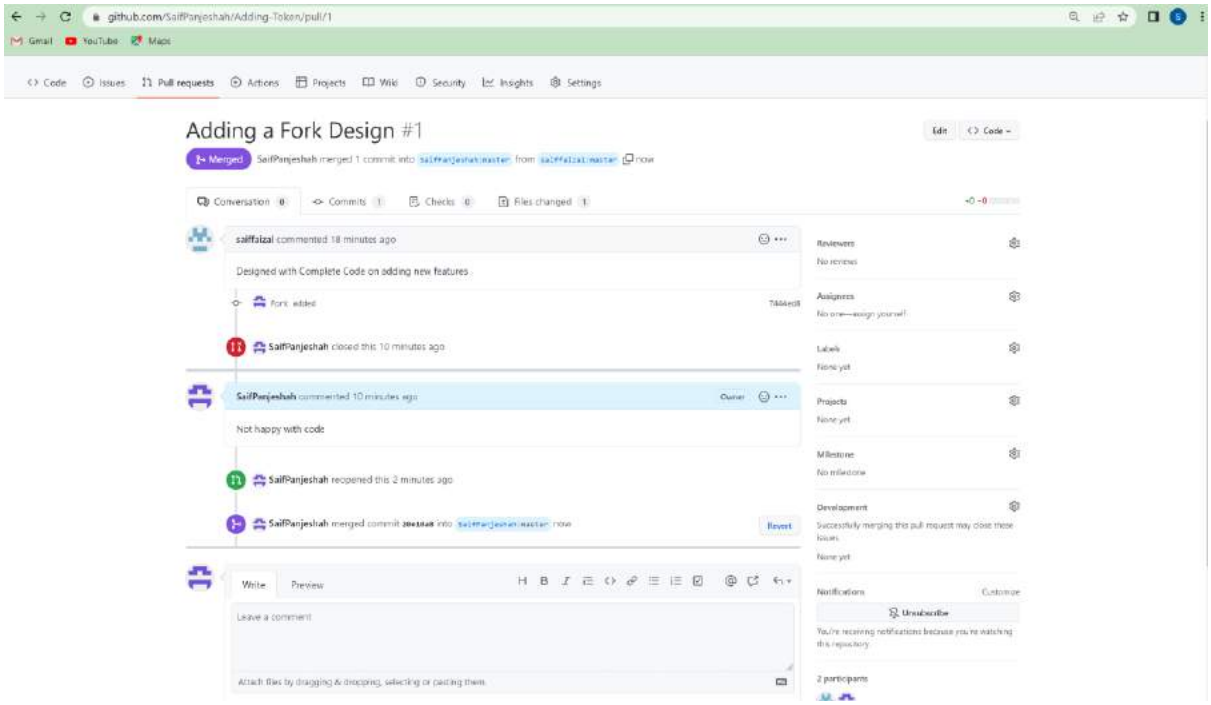


Fig. Merge successful

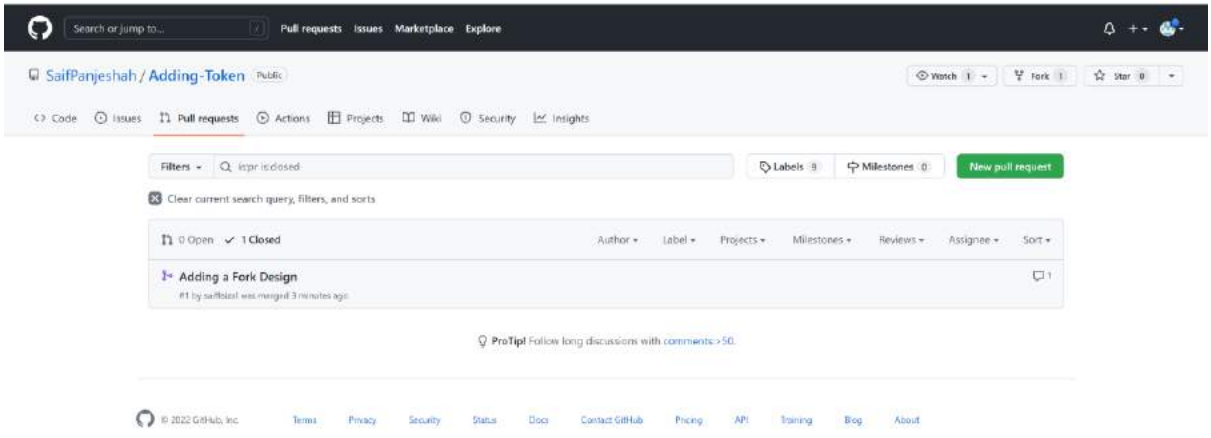


Fig. Close and successful Merge view from member account

Openings & Closing Issues:

Opening the Issue:

Creating an Issue from a repository:

1. **On GitHub.com, navigate to the main page of the repository Under your repository name, click on Issues ribbon.**

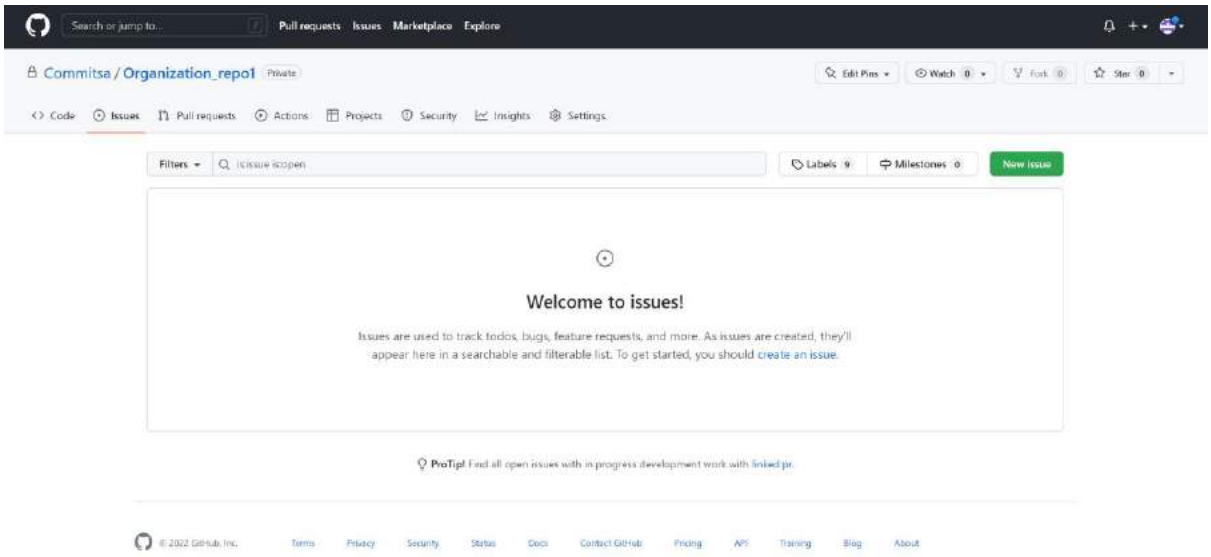


Fig. Creating a new Issue on organizational account repository

2. Click **New issue:**

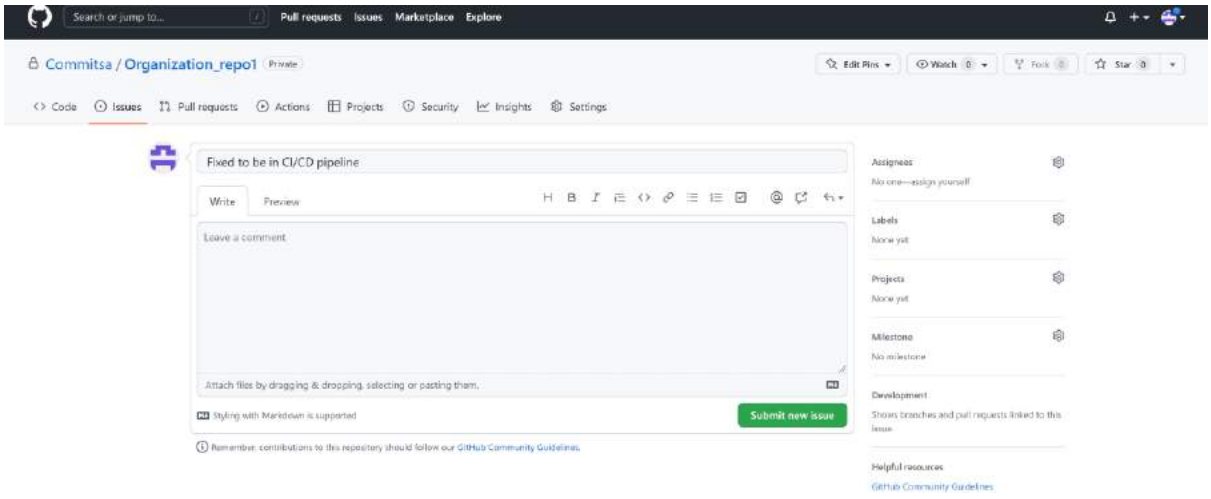


Fig. Creating Issues with CI/CD Pipelines of Devops

Lets assigned this to member user of this organization and add labels as bugs needed to be fix:

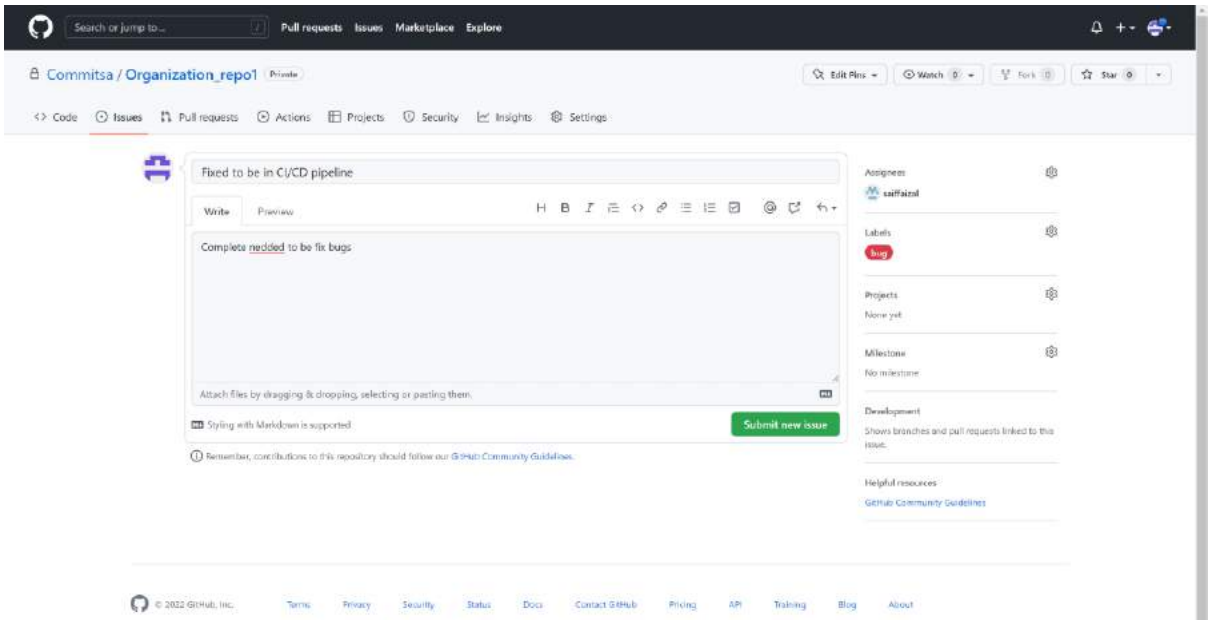


Fig. Fixed needs to be done

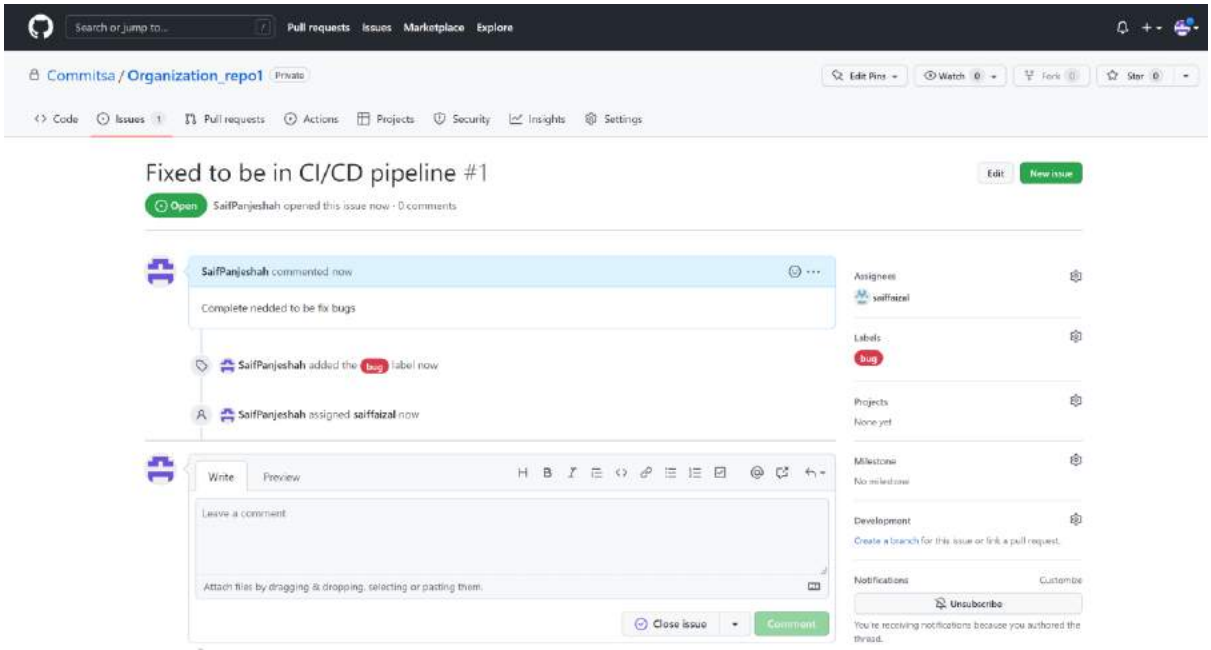


Fig. Issue successful Created

Closing the Issue:

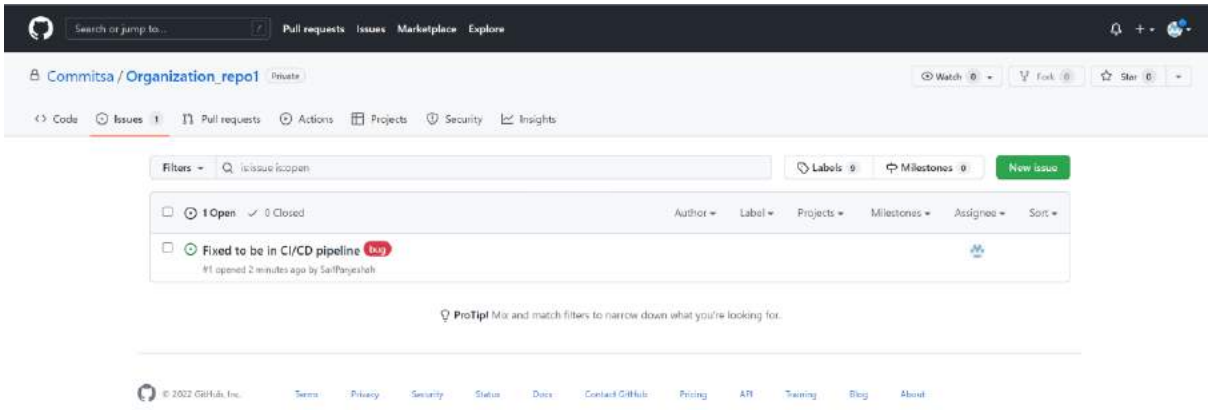


Fig. view on member account

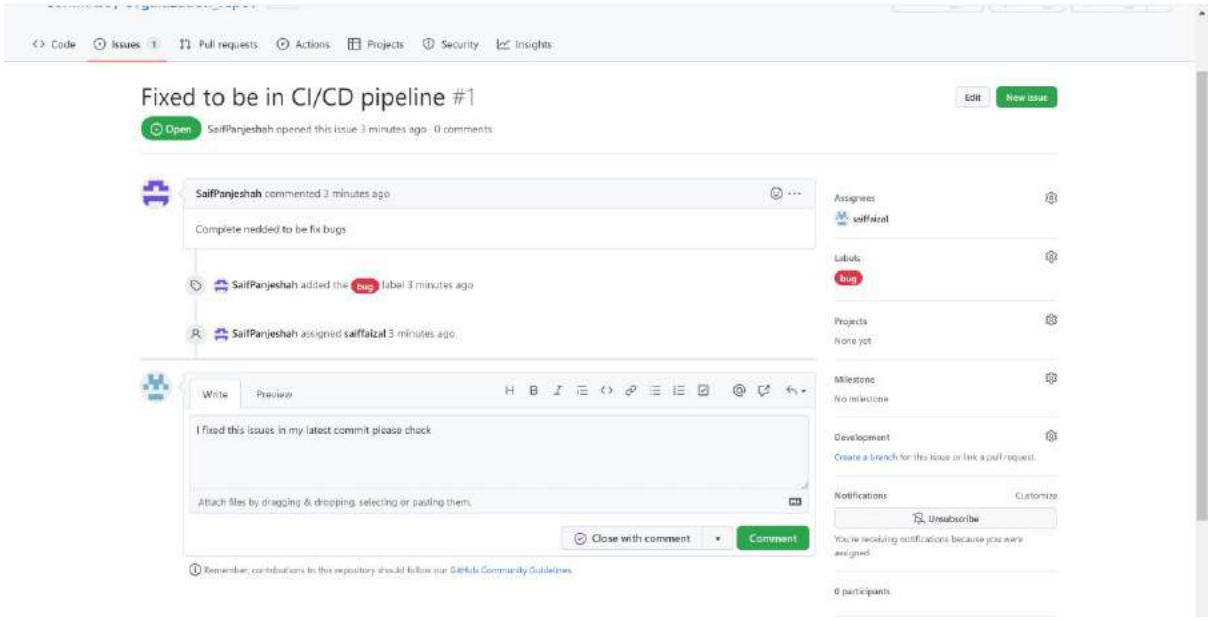


Fig. Member Fixed this issue

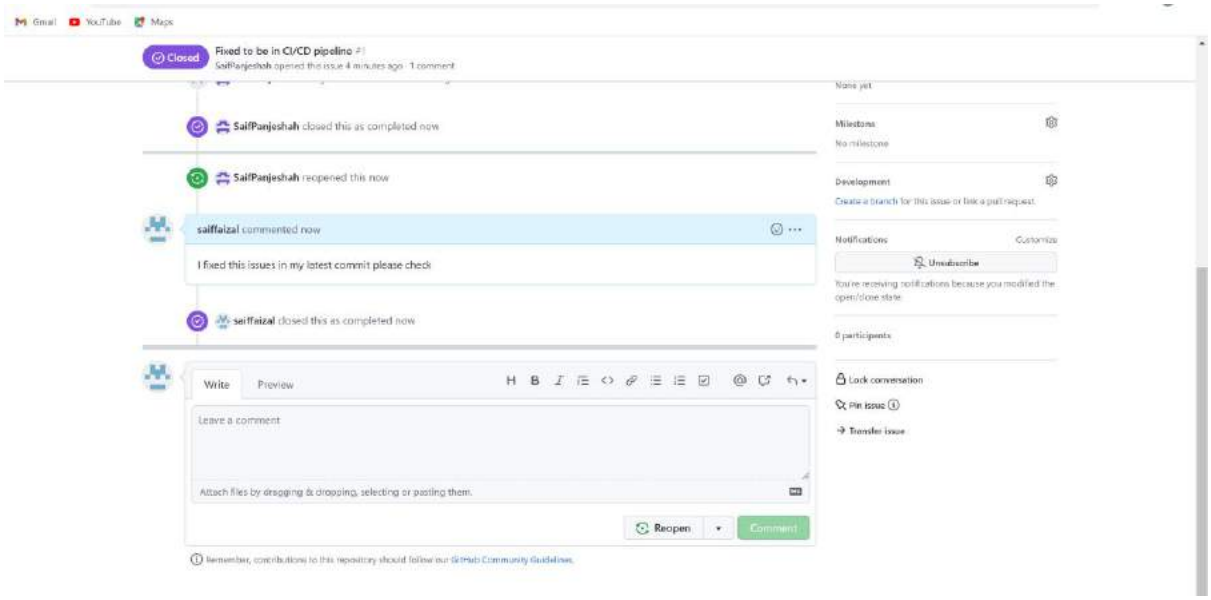


Fig. successful added close and fix commits

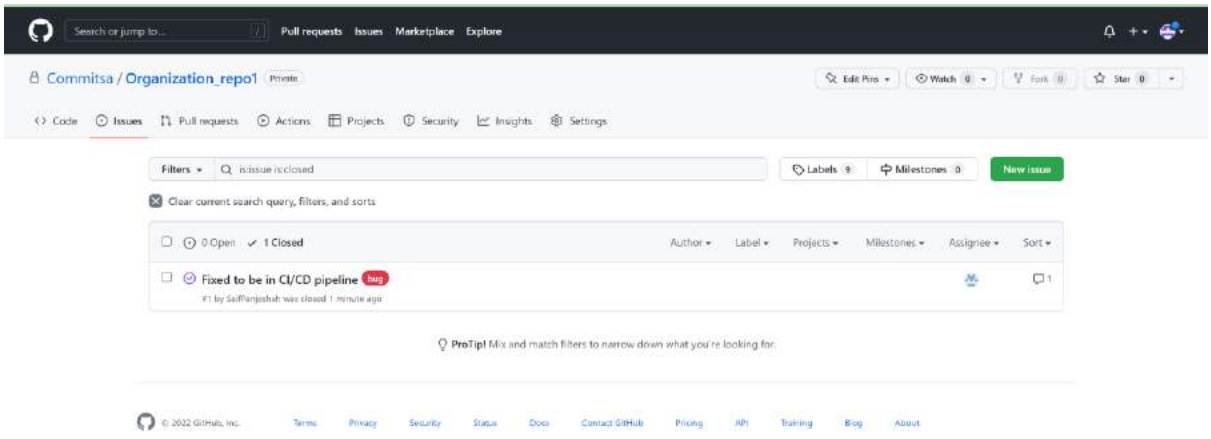


Fig. Successful close this Issue

Working with GitHub Projects:

Let's Explore the Project ribbon in GitHub:

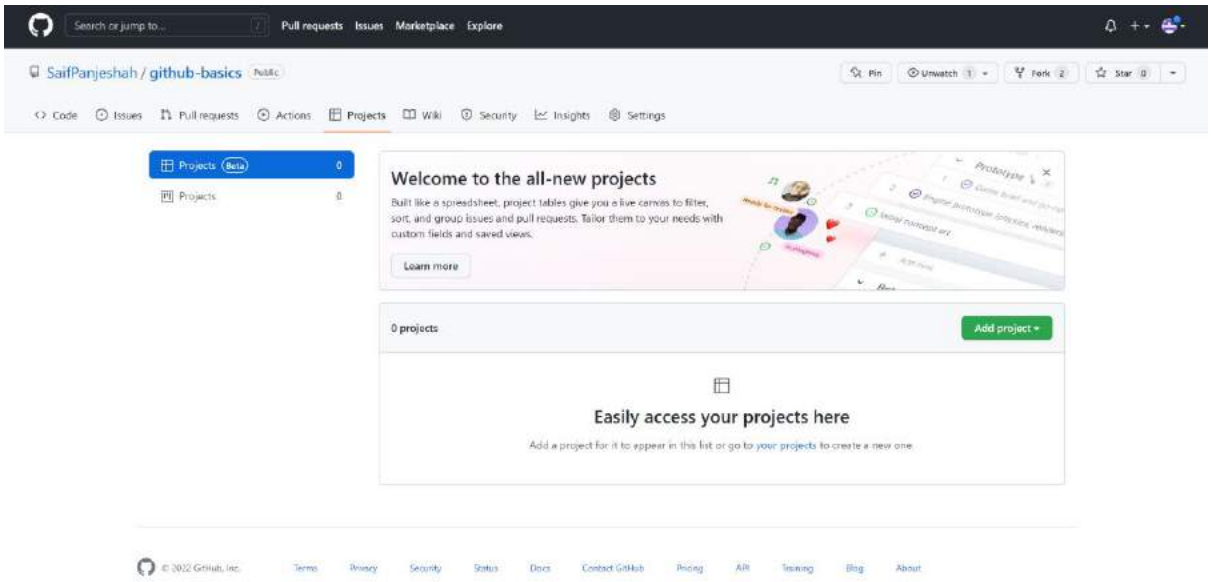


Fig. Project in GitHub

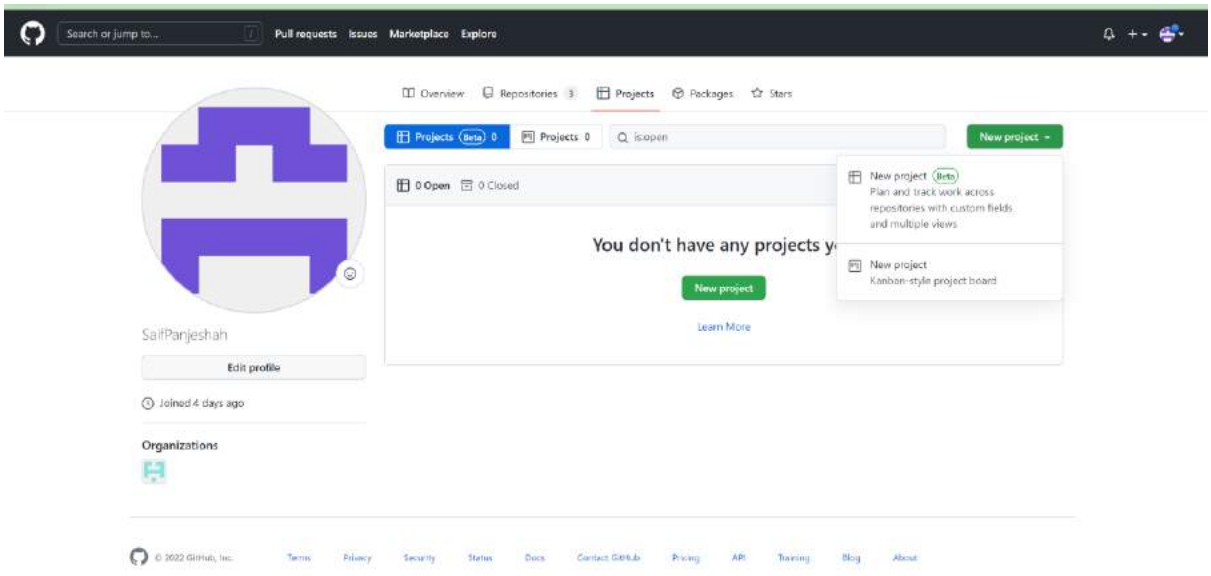


Fig. Creating a Project with Kanban Style

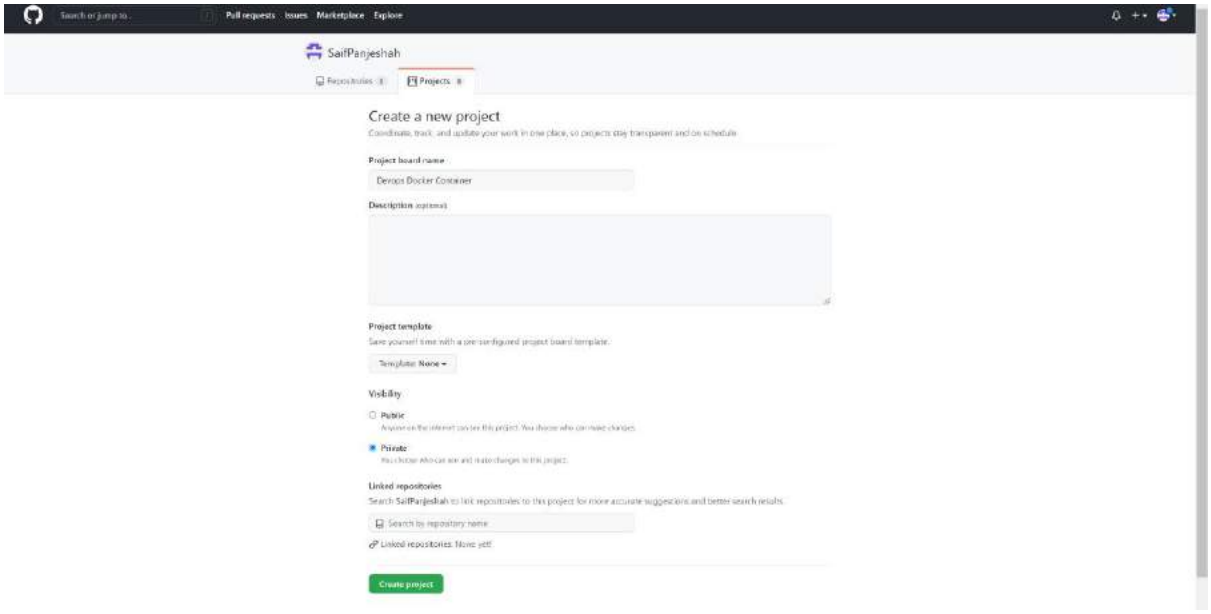


Fig. Devops Docker Container Project

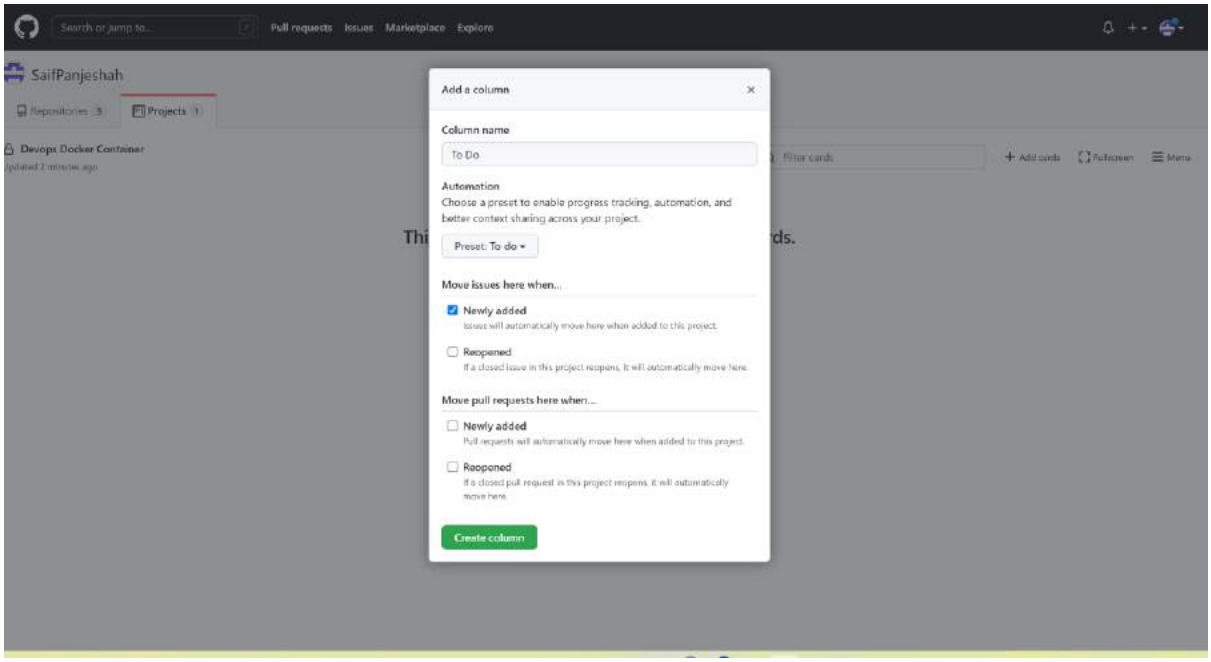


Fig. adding a column what to do

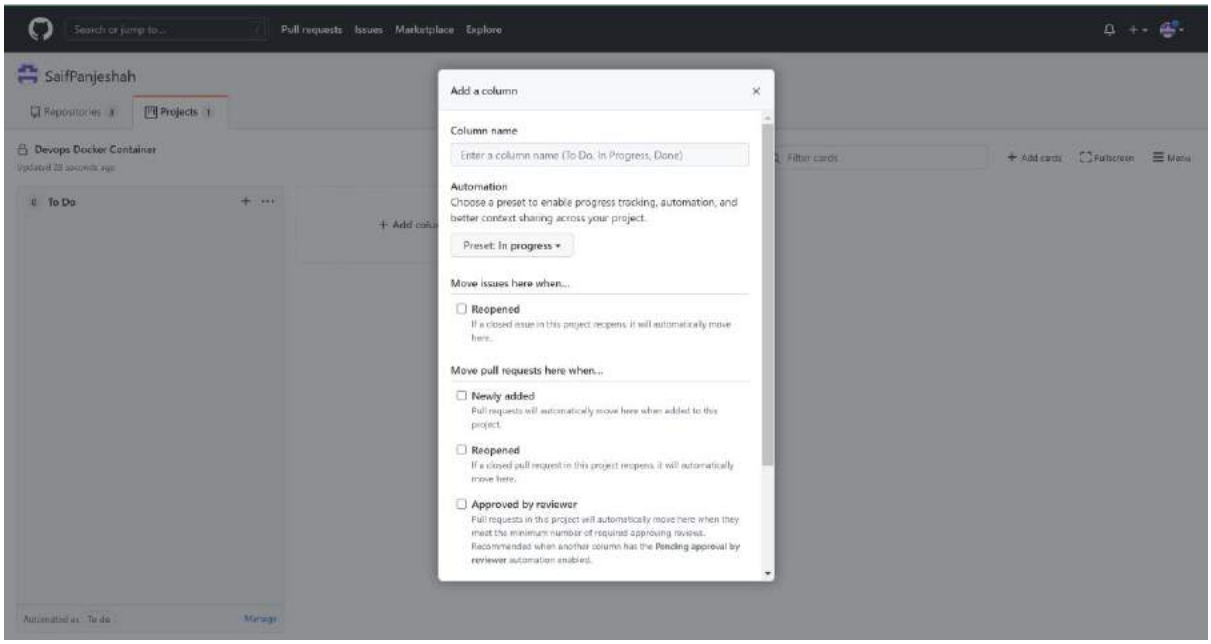


Fig. Creating another column with InProgress

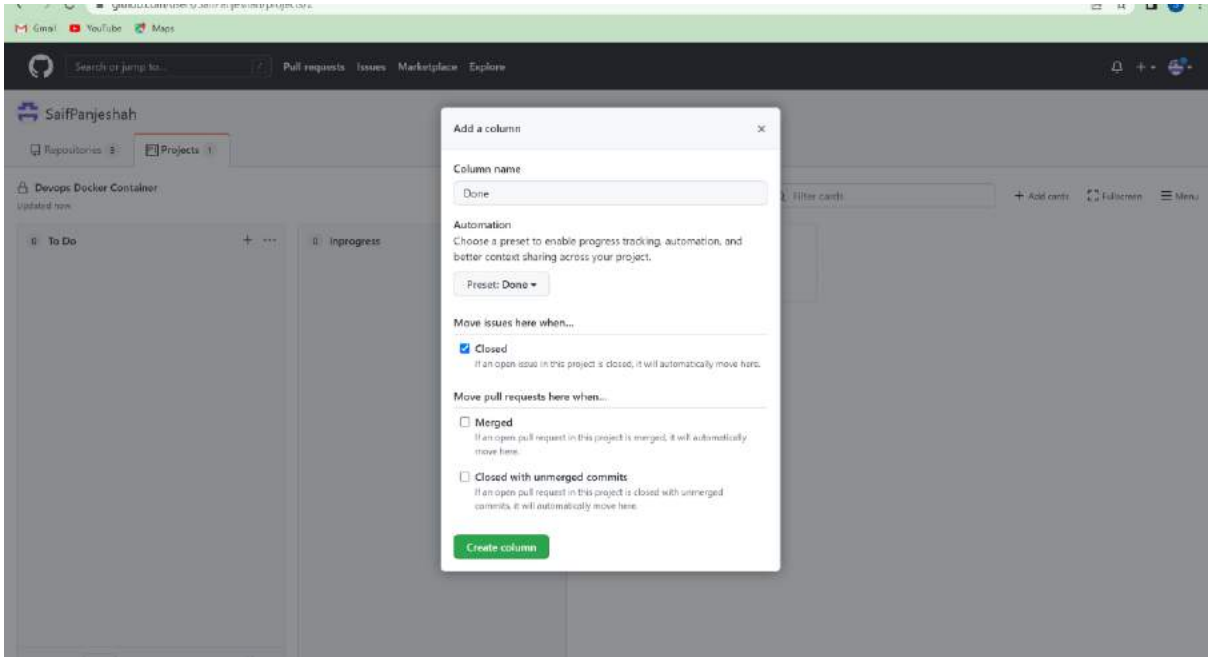


Fig. Final Column with work Done

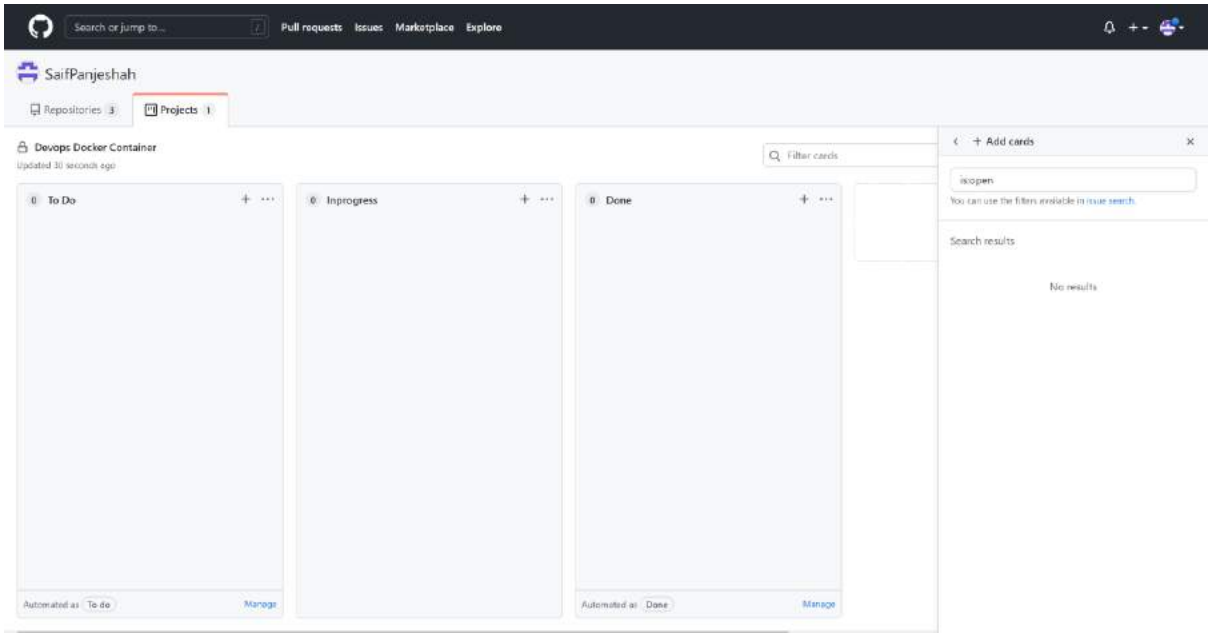


Fig. Project Created successful

Let's assign a work with open Issues on this Project:

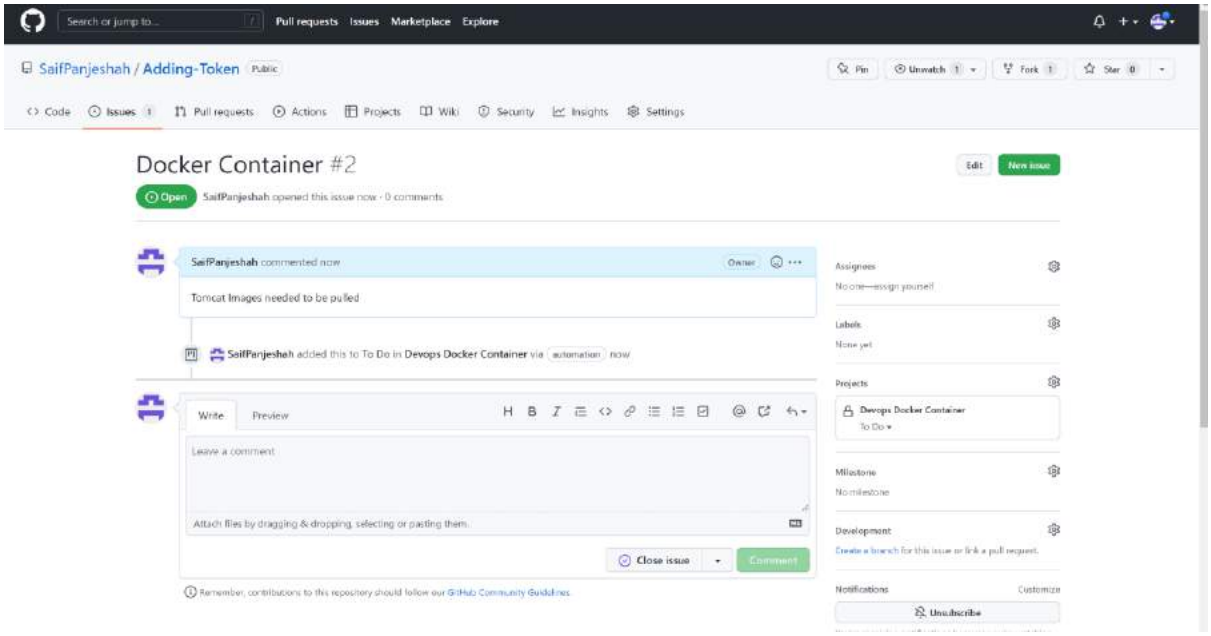


Fig. Creating new Issues

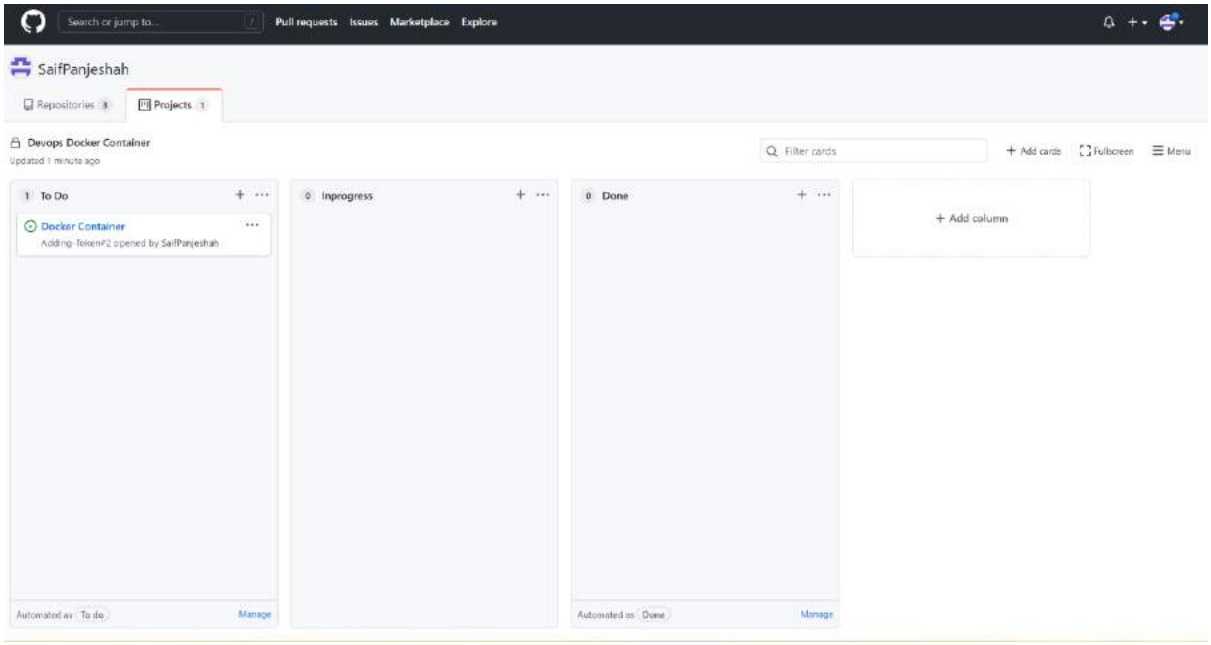


Fig. Card has been successful added to projects

Close this Issue and create a new Issues Check again the Devops Docker Container Project

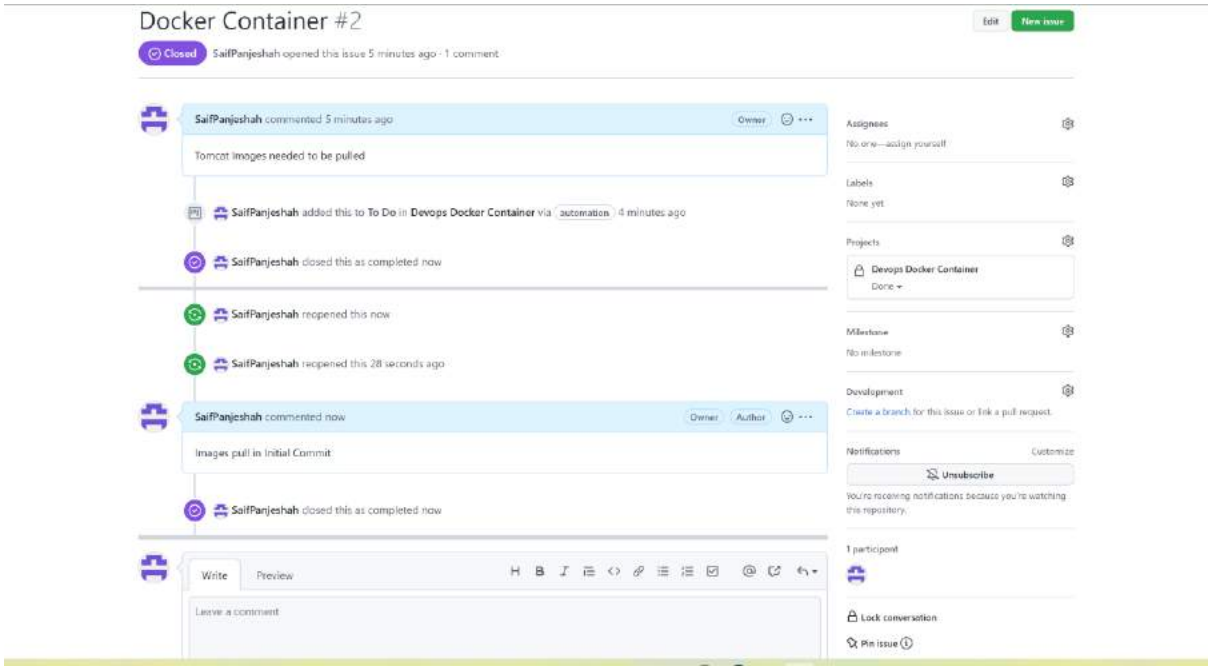


Fig. Closing Issue

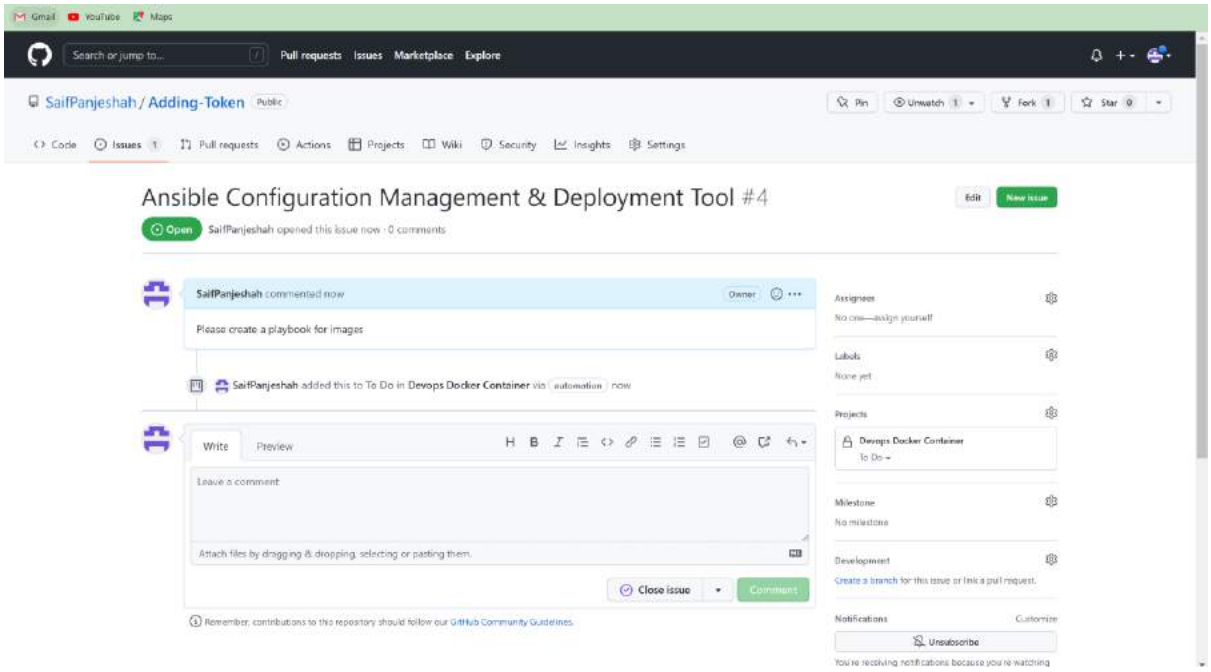


Fig. Creating New Issue

Let's Explore Devops Docker Container Project:

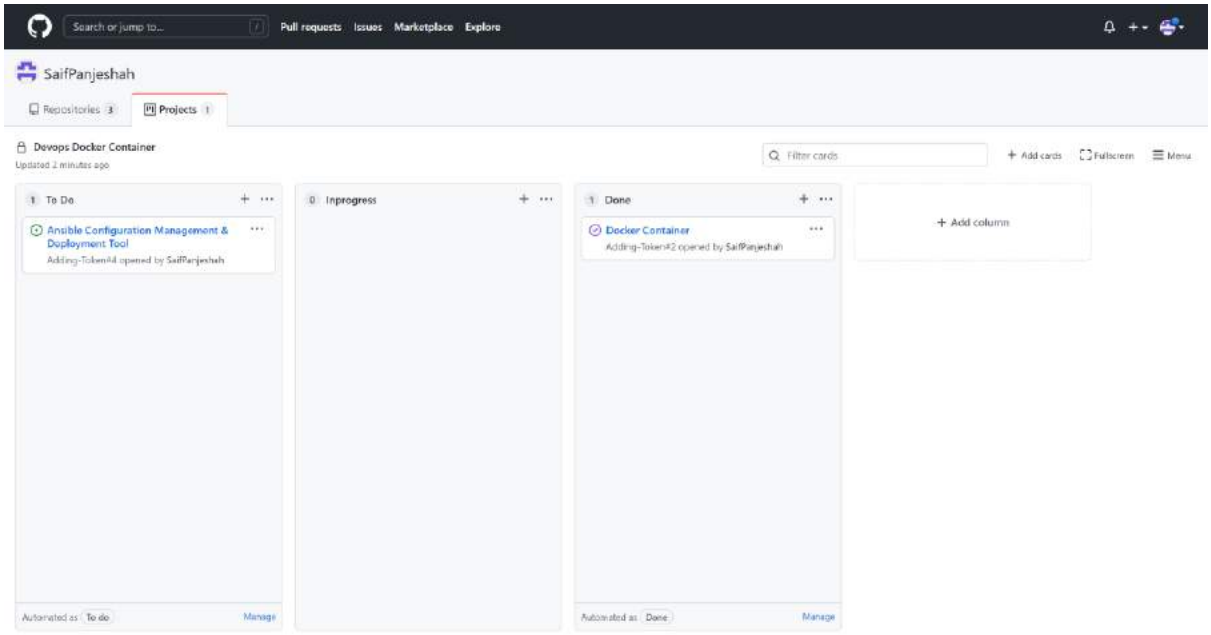


Fig. cards successful created on Devops Docker Container Project

Creating a README File:

README (as the name suggests: "read me") is the first file one should read when starting a new project. It's a set of useful information about a project, and a kind of manual. A README text file appears in many various places and refers not only to programming.

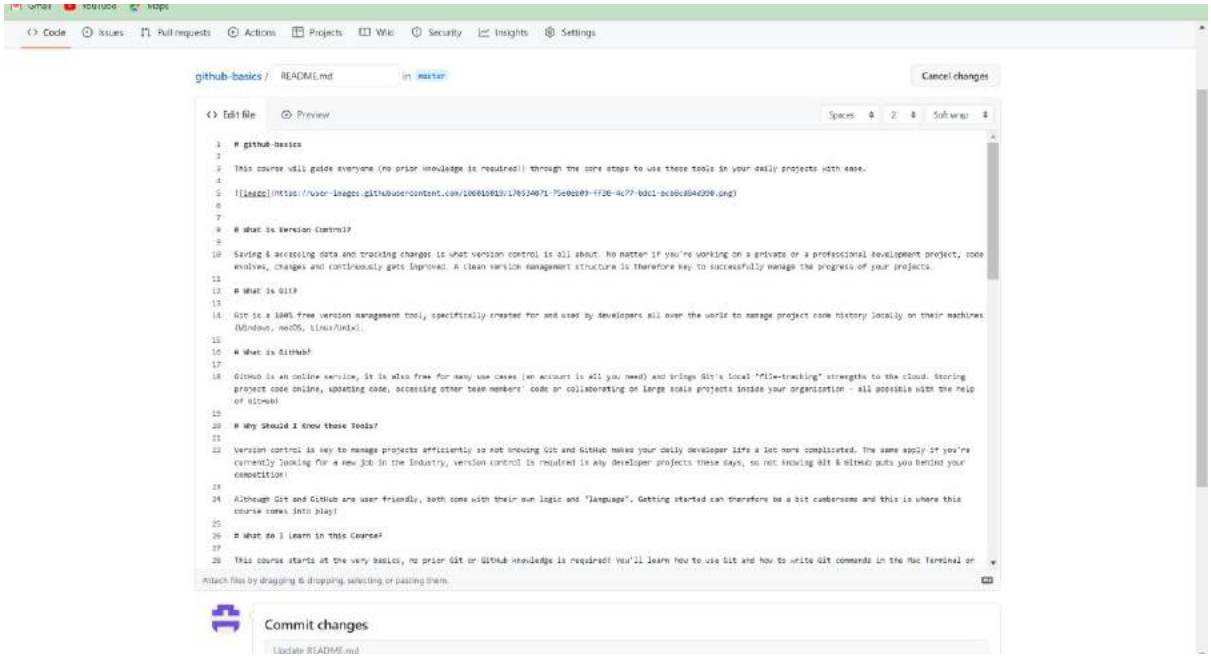


Fig. Creating a README File

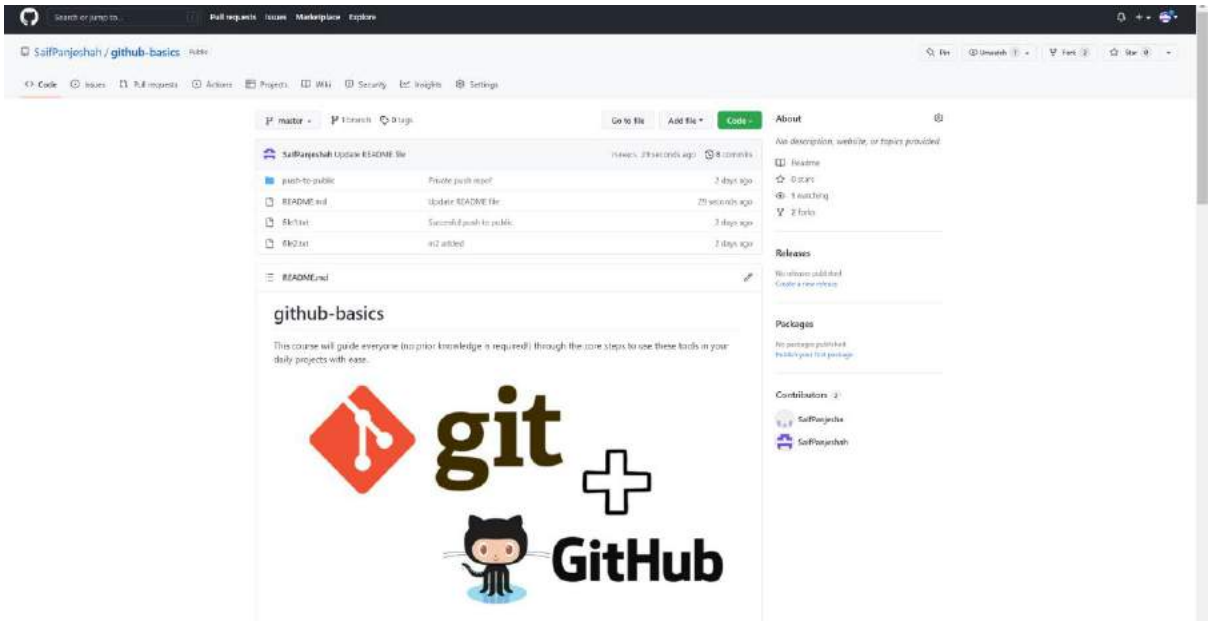


Fig. Complete README File added

Presenting Yourself as a Developer on GitHub:

Making GitHub Page more attractive.

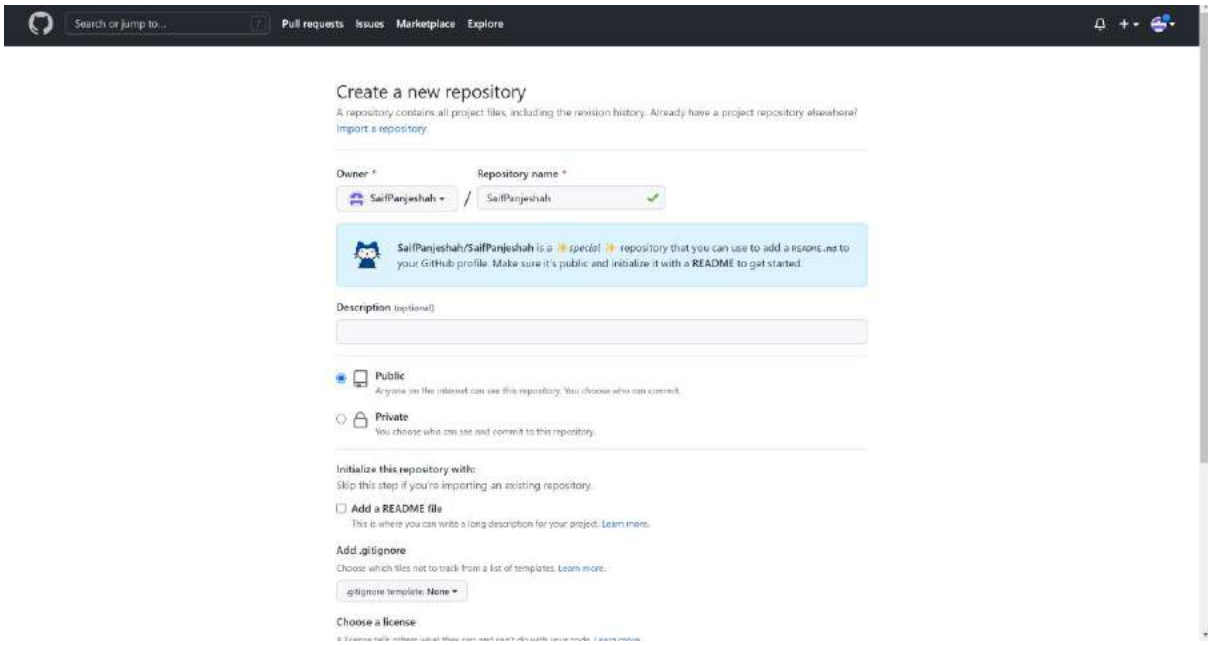


Fig. Creating Special secret repository with owner name

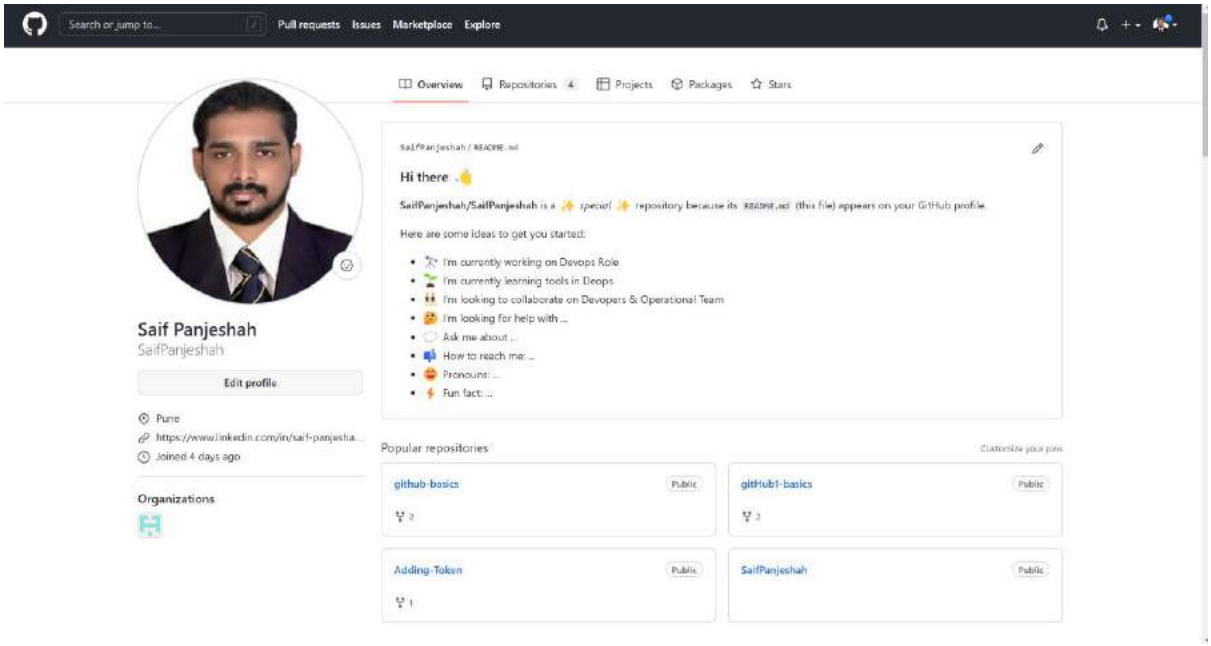


Fig. Presenting Yourself as a Developer on GitHub

About GitHub Stars:

Stars are all abouts likes on social media “Git and GitHub Basics “.

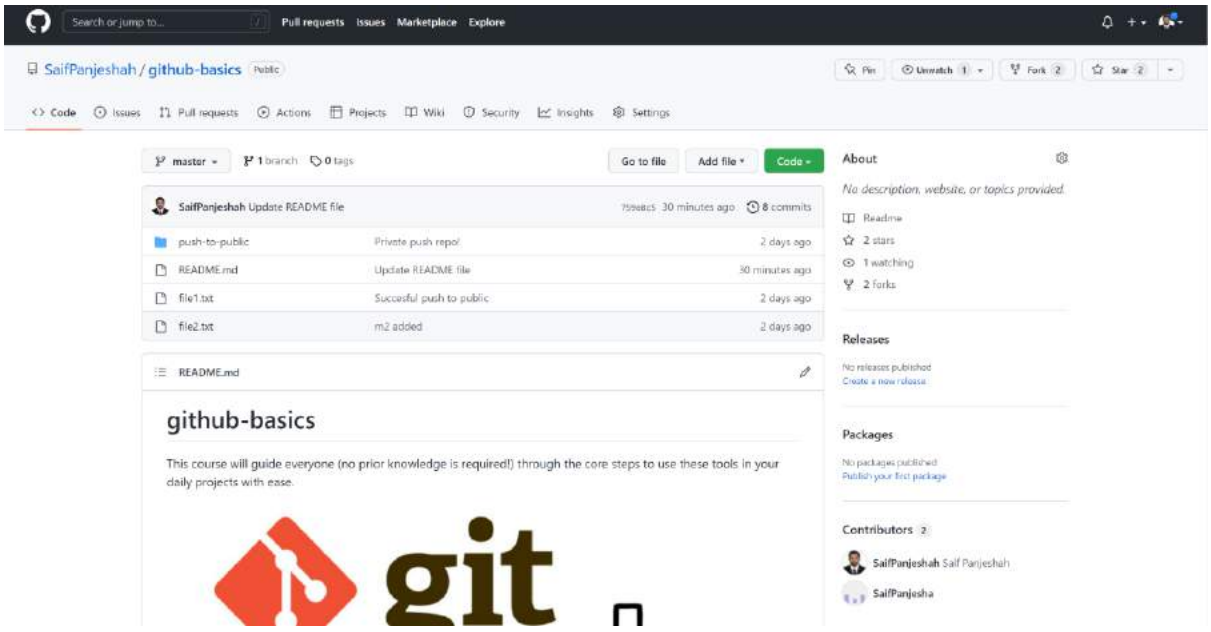


Fig. GitHub Stars

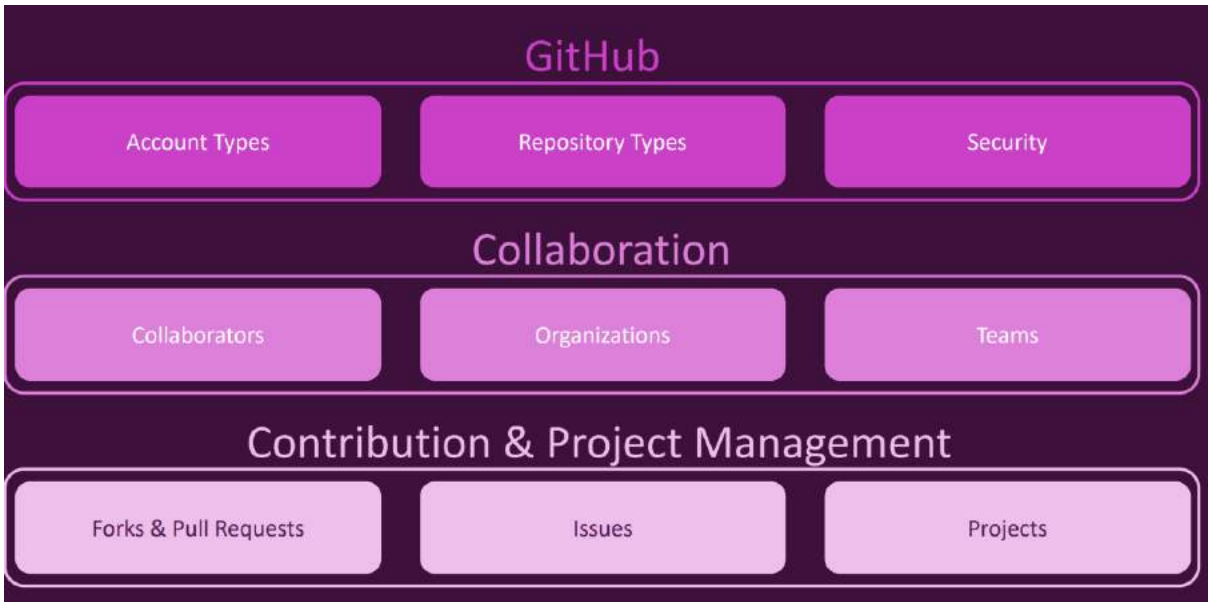


Fig. Wrap up Model Summary

Useful Resources and Link:

More about Permission Levels for User Account Repositories
=> https://docs.github.com/en/github/setting-up-and-managing-your-github-user-account/managing-user-account-settings/permission-levels-for-a-user-account-repository

VueJS GitHub Page => https://github.com/vuejs/vue