**Branching concept in GitHub**

**Branching** is the process of creating a diverging version of a codebase to develop features, fix bugs, or experiment without affecting the main project. It allows developers to work on different tasks simultaneously while keeping the main code stable. Once the changes are complete and tested, the branch can be merged back into the main codebase.

Or

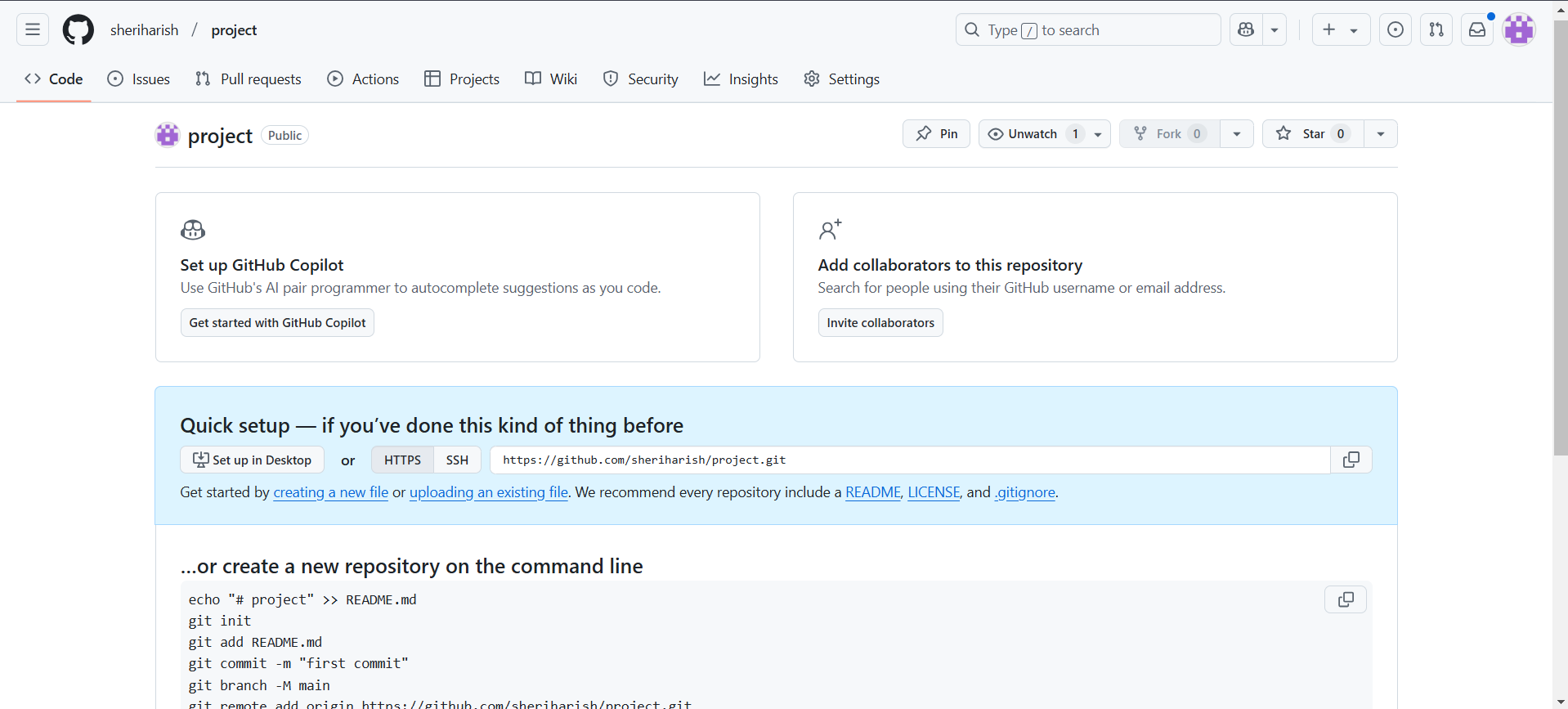
Branching in version control systems like Git (and platforms like GitHub) refers to the practice of creating separate lines of development within a repository. Each branch is an independent version of the codebase, allowing developers to work on new features, bug fixes, or experiments without affecting the main codebase (often called the **main**or **master** branch).

In Git, and by extension on platforms like GitHub, branching is a fundamental concept that enables parallel development and organized code management. Here's a breakdown:

**Why is Branching Important?**

1. **Feature Development:**
   1. Developers create branches to work on new features, ensuring that unfinished or unstable code doesn't disrupt the main codebase.
2. **Bug Fixes:**
   1. When a bug is discovered, a branch can be created to isolate the fix, test it thoroughly, and then merge it back into the main branch.
3. **Experimentation:**
   1. Branches allow for safe experimentation with new ideas or code changes without risking the stability of the main codebase.
4. **Version Control:**
   1. Branching helps maintain different versions of the code, making it easier to manage releases and roll back changes if necessary.
5. **Collaboration**:
   1. Multiple developers can work on different branches simultaneously, enabling parallel development.

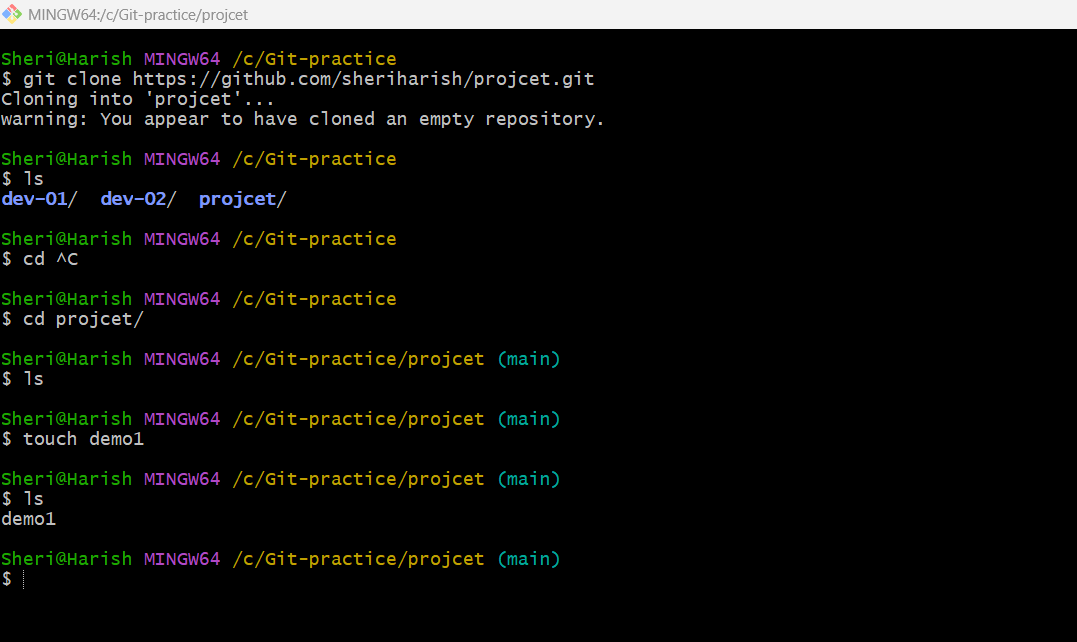
Let’s do branching concept practically.

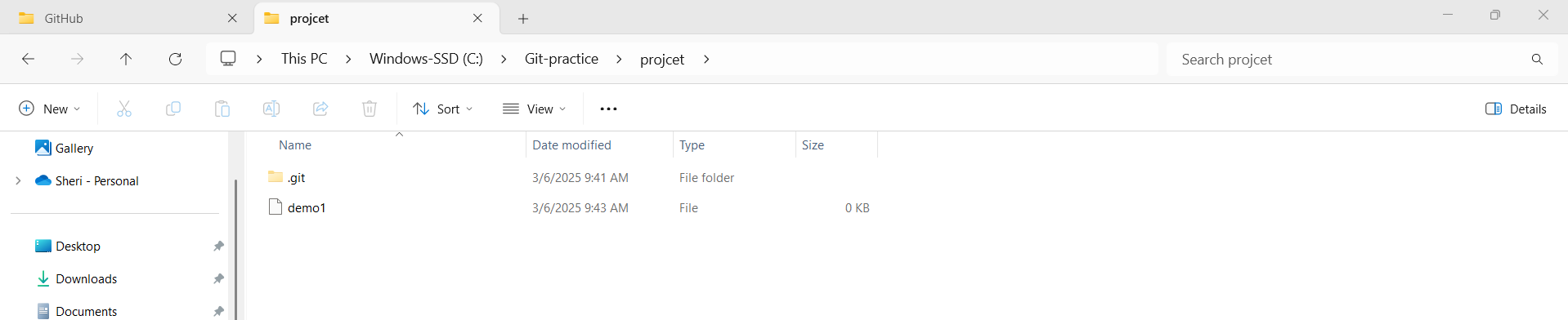
**Step1**: Create a repository (project) in GitHub.

**Note1:** If you create a new repository from GitHub website then the name of branch is by default “**main”.**

**Note2:** when you initialize a new Git repository from locally machine using the <git init> command, Git's historical default branch name has been **"master”.**

**Step2:** clone the repository (project) to the local machine and create a file (demo1) in main branch.



Fig: file (demo1) is created.

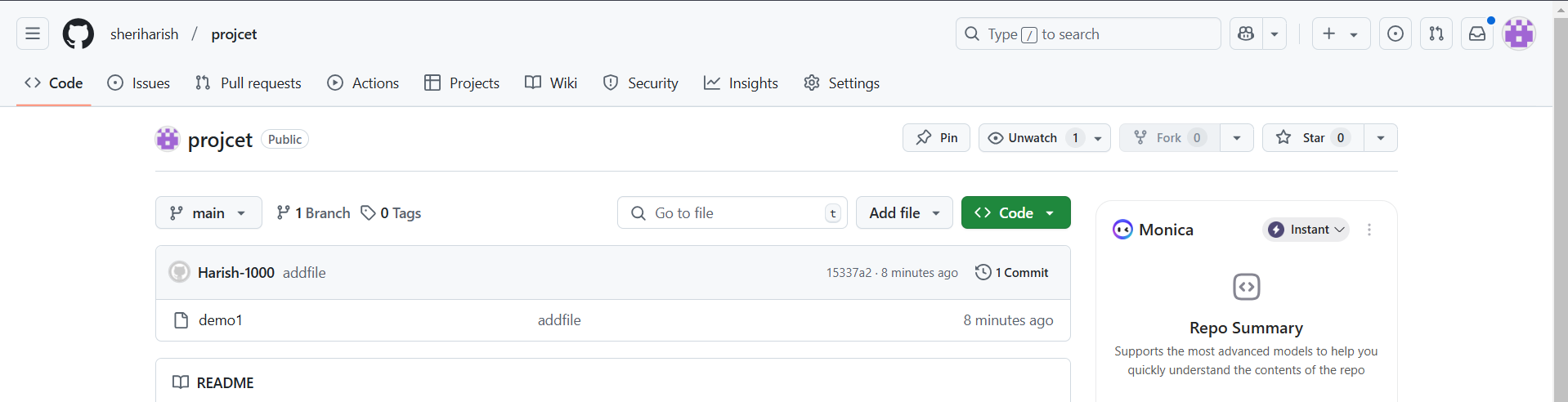
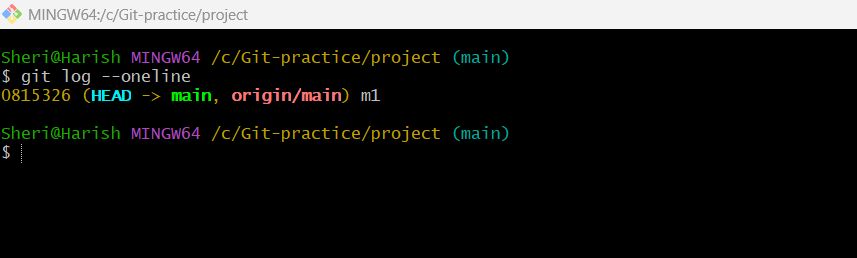


Fig: GitHub GUI view.



**Fig:** Main branch commit status.

The branch **main** has only single commit (m1) as shown in above figure.

**Step3:** Create another branch (development) and within it create a file (demo2).

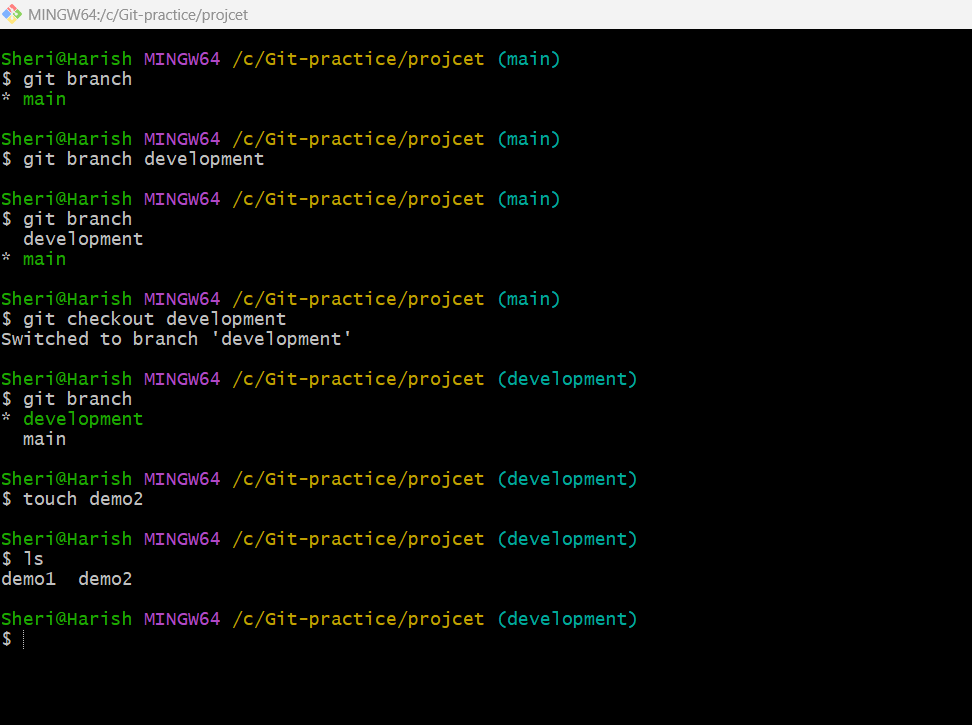
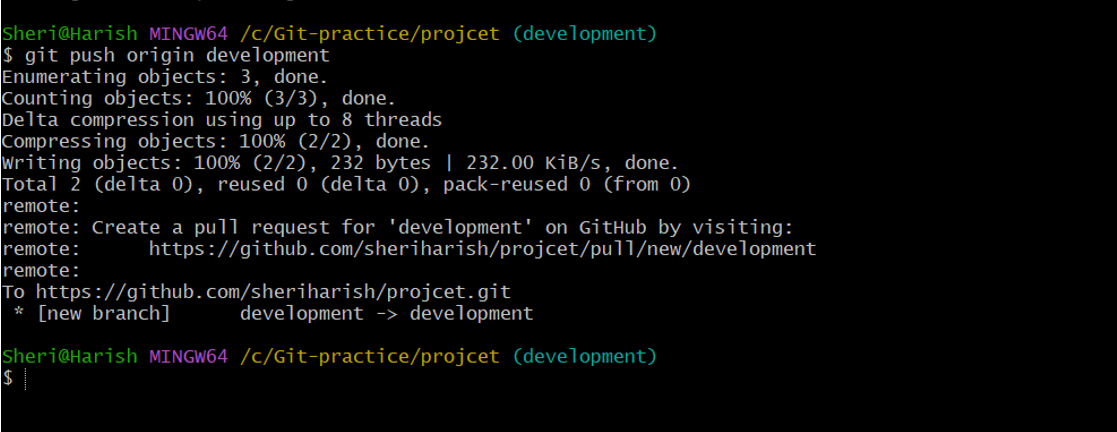


Fig: New branch (development) & file (demo2) is created successfully.

**git branch**: shows the list of branches in current repository.

**git branch development:** Used to create the new branch with “development”.

**git checkout development**: Used to switch the branch from **main** to **development** branch.

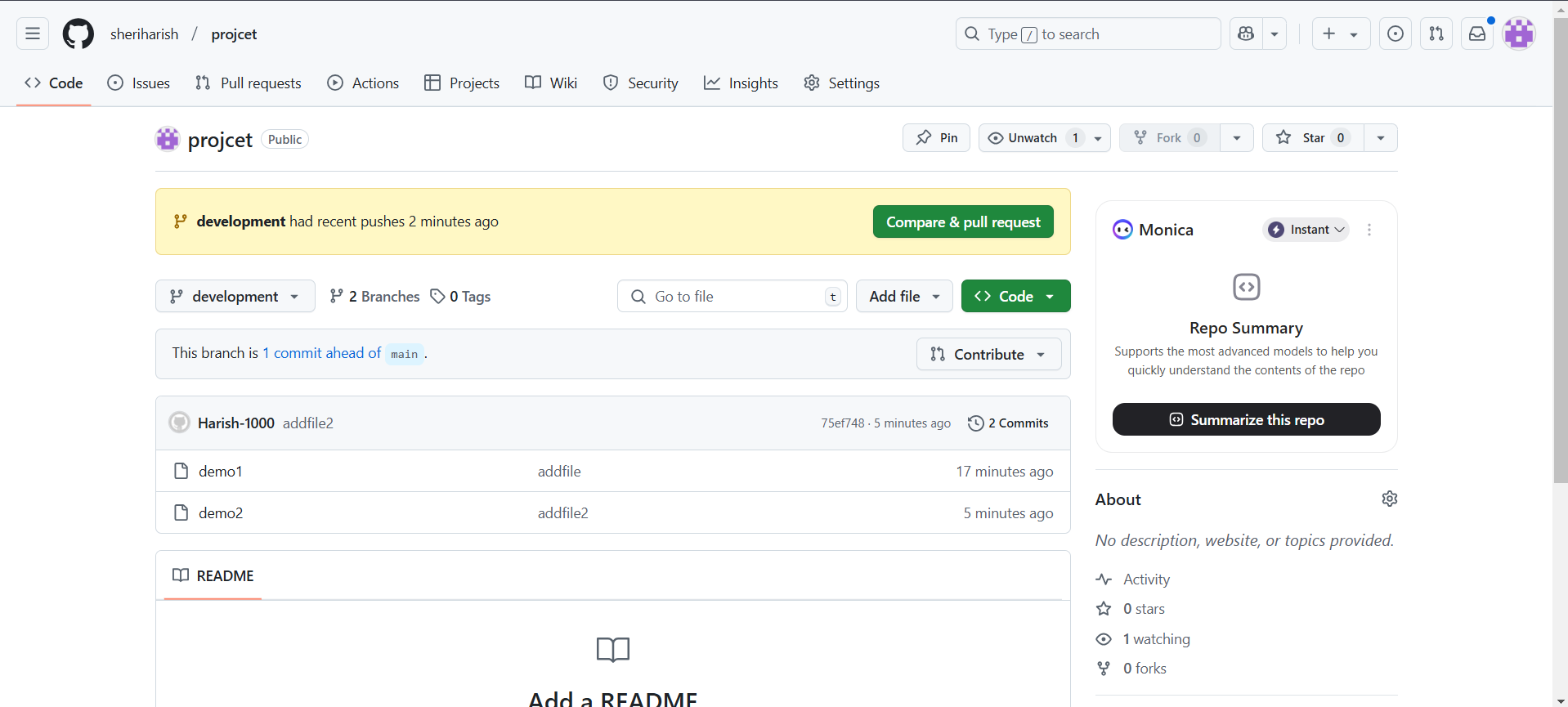


**Note:** In order pushing the file to new branch we should use the command **git push origin <branchname>.**

****

**Fig:** commit status of development branch.

The branch development has two commits (m1 & m2) as show above figure.

****It is because while creating a child branch under the main branch it will copy the commits as well as files (demo1) from main branch to the child branch (development).

Main branch

Demo1

m1

Development branch

Demo1, demo2

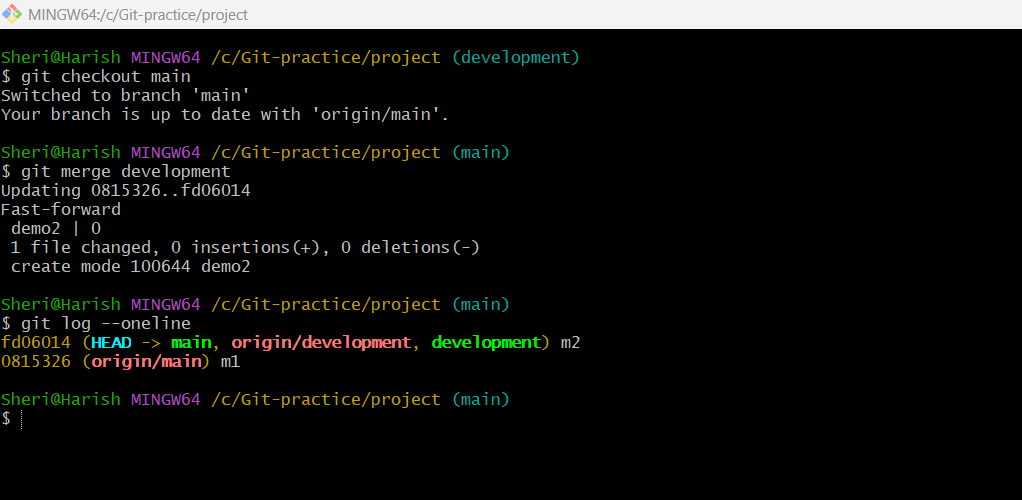
m1 & m2

**Fig:** Block diagram of above task.

**Step4:** Now merge both the branches (main & development).

process of branch merge:

1. **Switch to the Target Branch:**
   * First, you need to switch to the branch that you want to merge *into*. This is often your "**main**" or "**master"** branch.
   * Use the command: git checkout <target-branch-name>
     + **Example:** git checkout main
2. **Perform the Merge:**
   * Now, use the git merge command, specifying the branch that you want to merge *from*.
   * Use the command: git merge <source-branch-name>
     + **Example**: git merge feature-branch
3. **Handle Conflicts (If Any):**
   * Sometimes, Git might encounter conflicts if the same lines of code were changed in both branches.
   * If this happens:
     + Git will mark the conflicting areas in the affected files.
     + You'll need to manually edit those files to resolve the conflicts.
     + Then, use “**git add**” to stage the resolved files.
     + Finally, use “**git commit”** to complete the merge.
4. **Complete the Merge:**
   * If there are no conflicts, or once you have resolved all the conflicts, Git will create a new "merge commit" that combines the changes.



In order to merge the development branch with main branch first we have to stand on the path of main branch then do **“git merge development”.** So that all the files and commit lists can be appeared in the main branch of git repository (project) as show in above figure.

GUI view of merging in GitHub website:

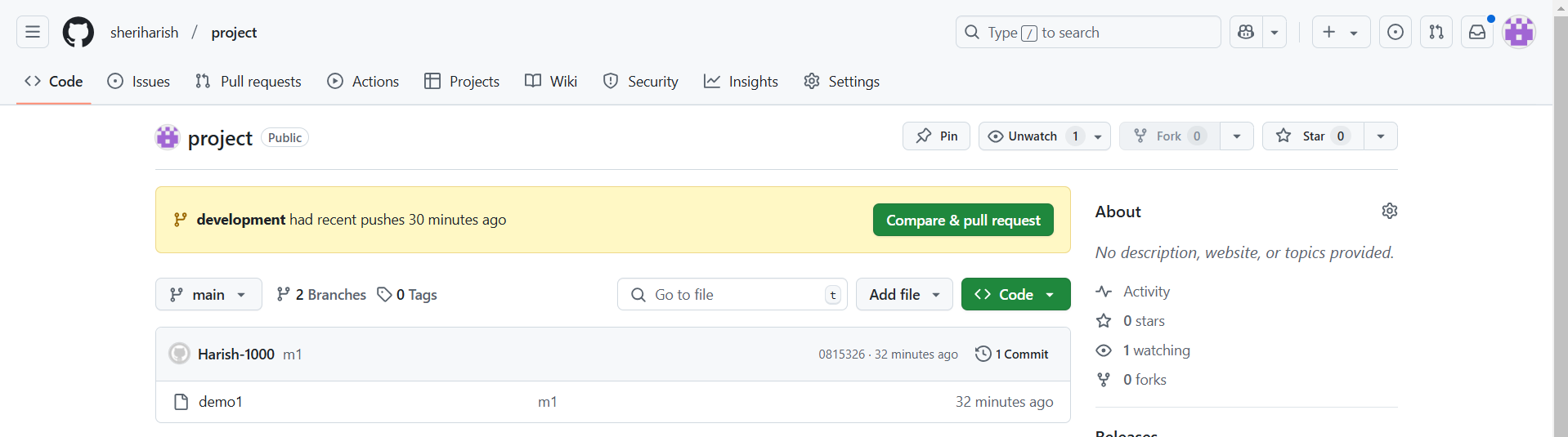
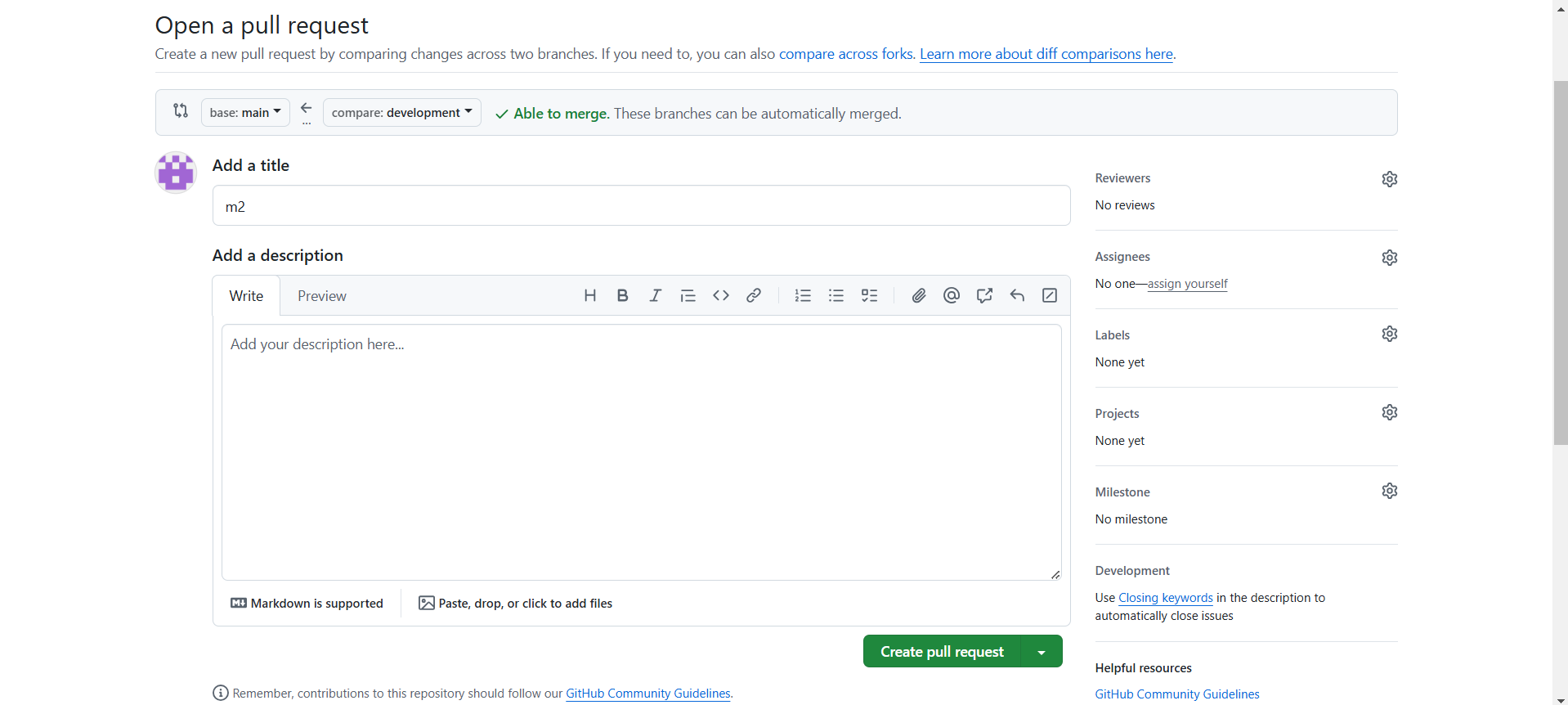


Fig: compare & pull request.



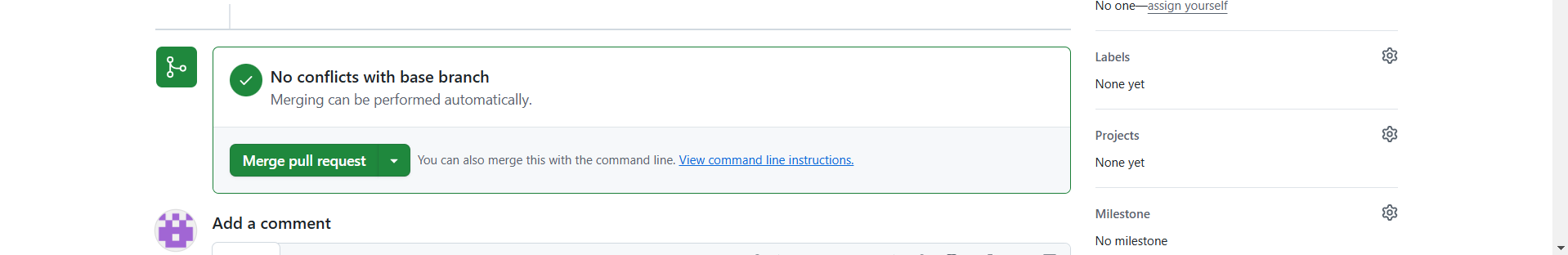
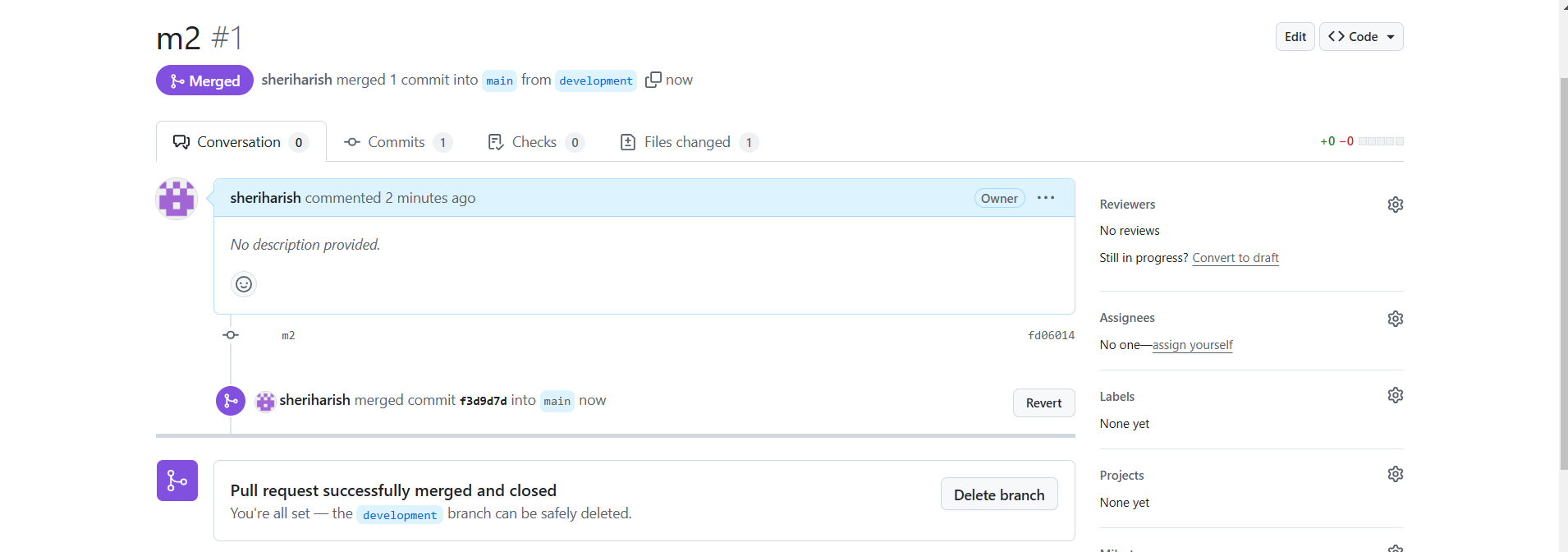
Fig: Create pull request

Fig: merge pull request.

Fig: Delete the branch (development) after merging is done if required.

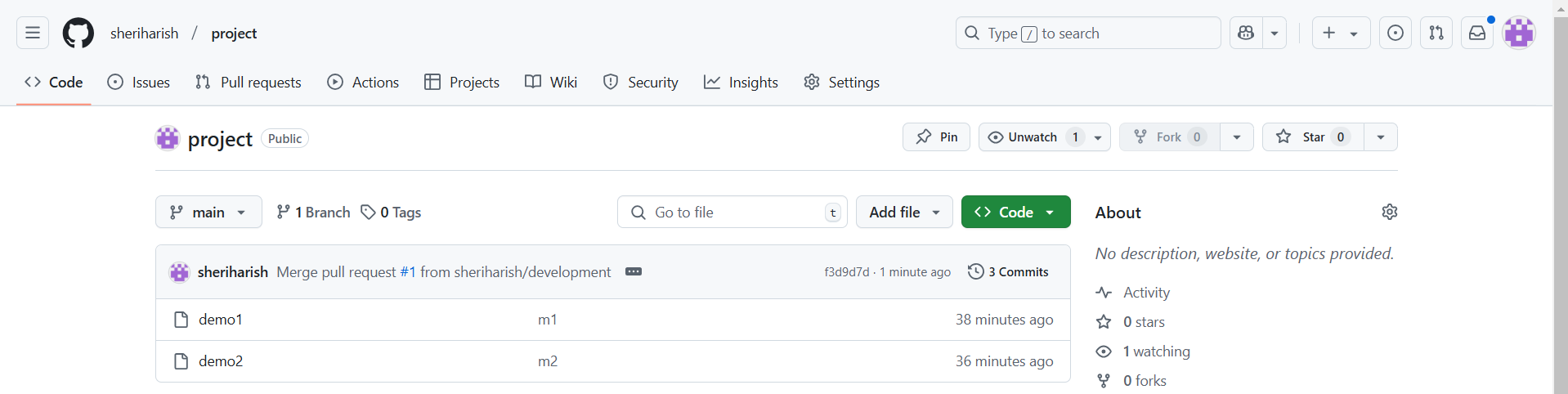


Fig: Two files demo1& demo2 are present in main branch after merging.