**MERGING CONCEPT**

In GitHub, **merge** refers to combining changes from one branch into another. This is a fundamental part of Git-based workflows, allowing multiple developers to collaborate and integrate their work.

Or

In Git, a **merge** is the process of combining changes from one branch into another. It is a fundamental operation in version control that allows developers to integrate work done in different branches. Merging is commonly used to bring feature branches into the main branch (e.g., main or master) or to synchronize changes between branches.

**Types of Merges in GitHub**

GitHub provides different types of merging strategies:

1. **Merge Commit (Default Merge)**
   * This creates a new merge commit that combines the history of both branches.
   * Preserves the commit history.
   * Command: **git merge branch-name**
2. **Squash and Merge**
   * Combines all commits from the source branch into a single commit before merging.
   * Keeps a cleaner commit history.
   * Useful for feature branches.
   * It is not recommended in real world.
   * Command: **git merge –squash branch-name** then do **git commit.**
3. **Rebase and Merge**
   * Moves the entire feature branch on top of the target branch, creating a linear history. That means it provides commit history in order wise.
   * Avoids merge commits.
   * Helps maintain a cleaner Git history.
4. **Cherry-pick**
   * It is used to merge/combined a specific commit from one branch (feature-branch) to another branch (main/master branch) without merging an entire branch.

Main

m2

m1

Main-Branch

m1

f1

Feature- Branch

Perform the above block diagram task for all merging concepts such as default merge, squash merge and rebase merge.

1. **Merge commit (default merge).**

Step1: Create a commit “m1” by creating file in main branch.

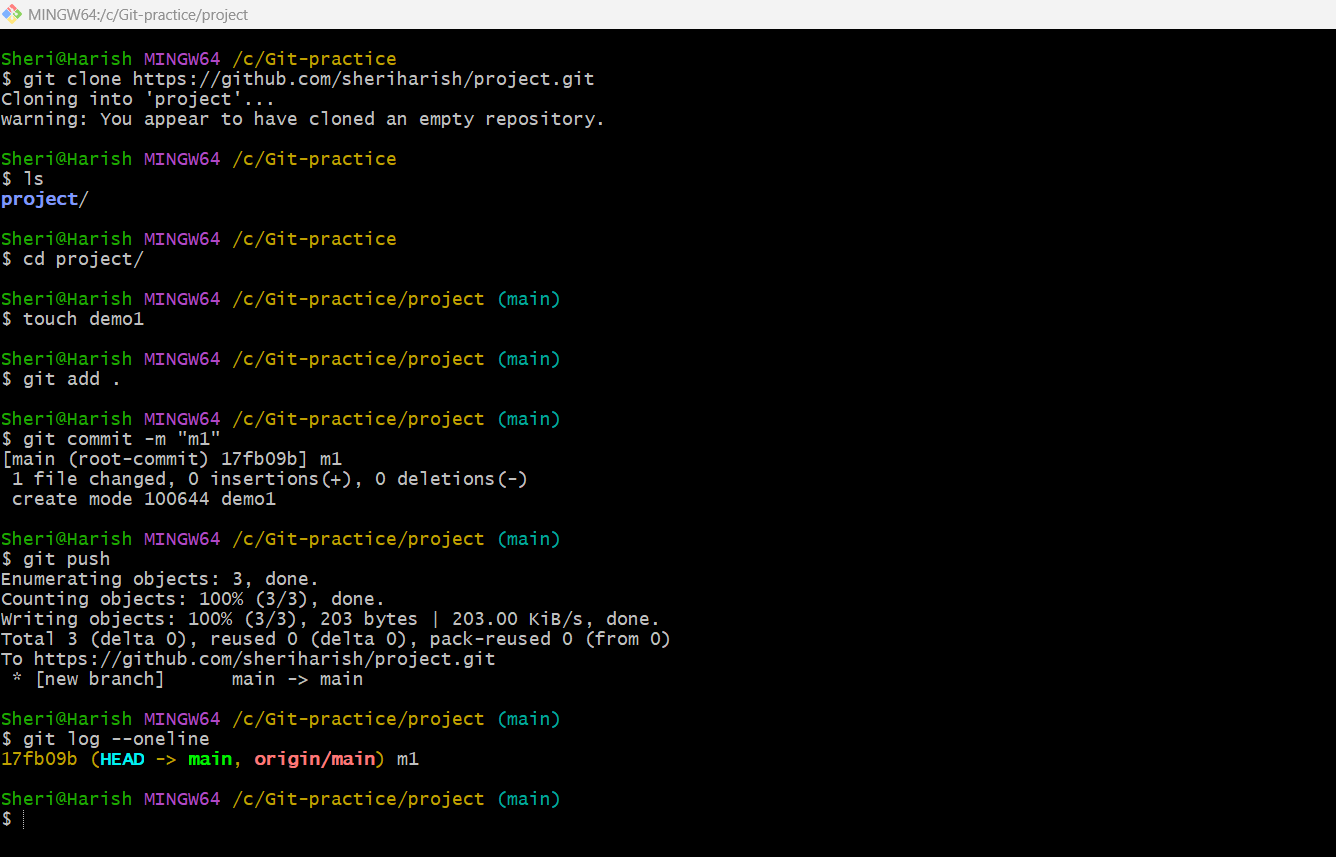
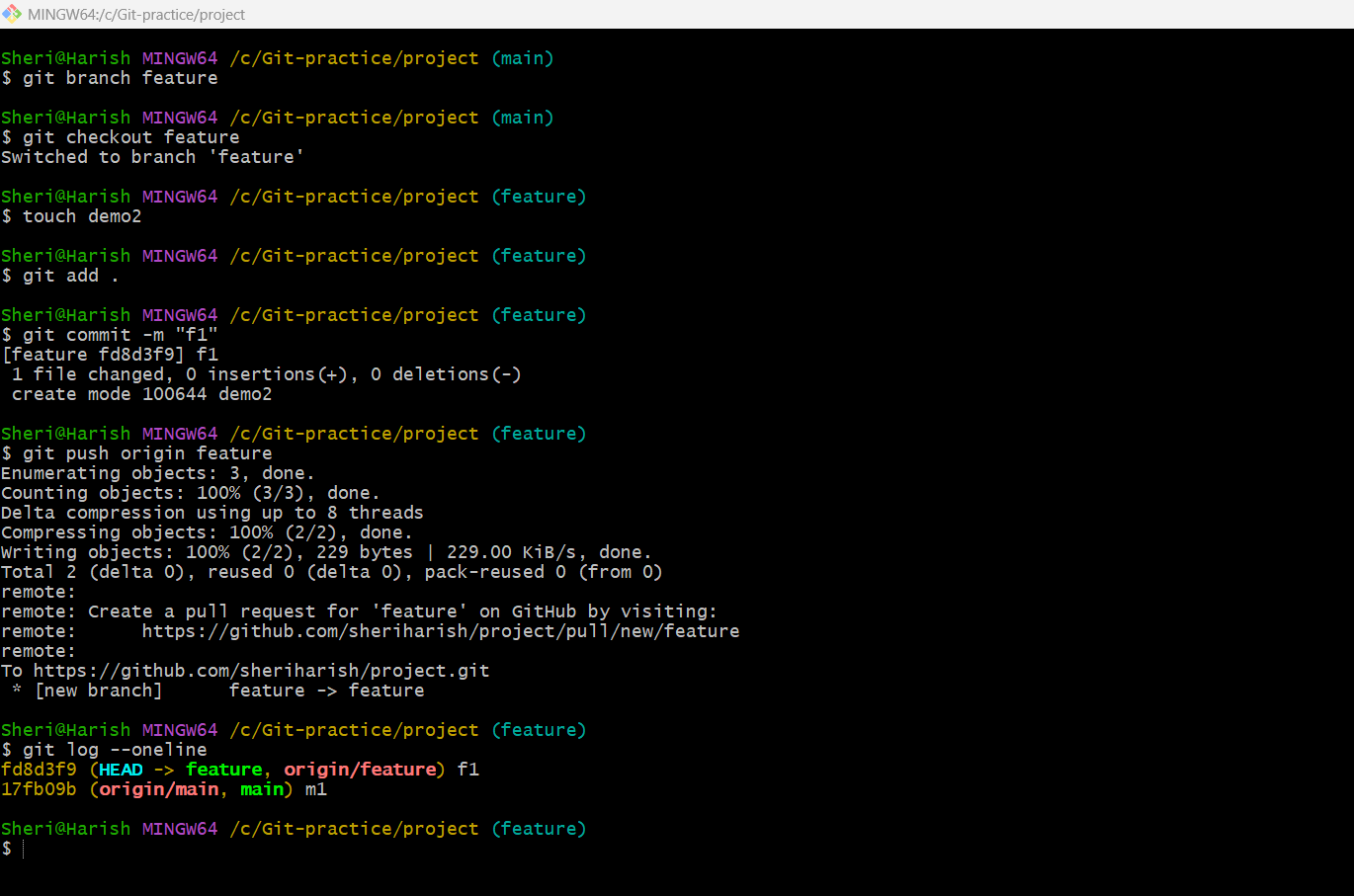
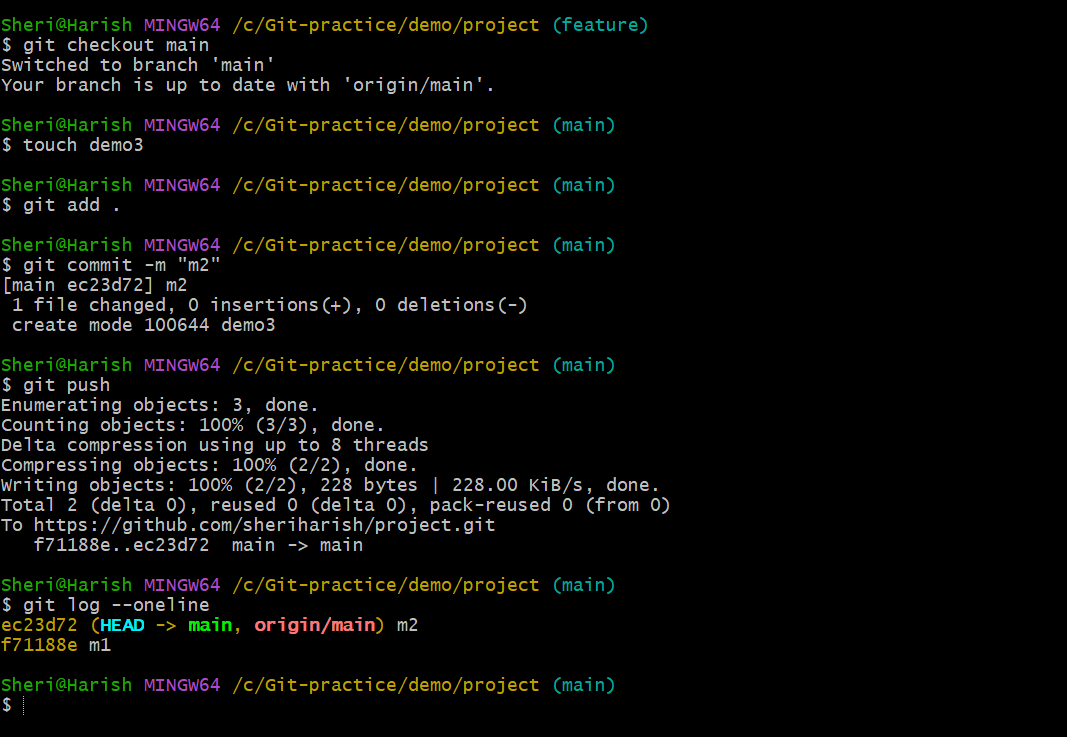


Fig: In main branch ‘m1’ commit is done by create a file demo1.

**Step2:** Create a new branch **feature** and create a file (demo2) and commit it (f1).

**Note:** Feature branch consists two commits m1 & f1 as show in below figure.

**Step3:** By switching to main branch again create another file and commit it as “m2”.



**Step4:** Now do the default merge <git merge branch-name>.

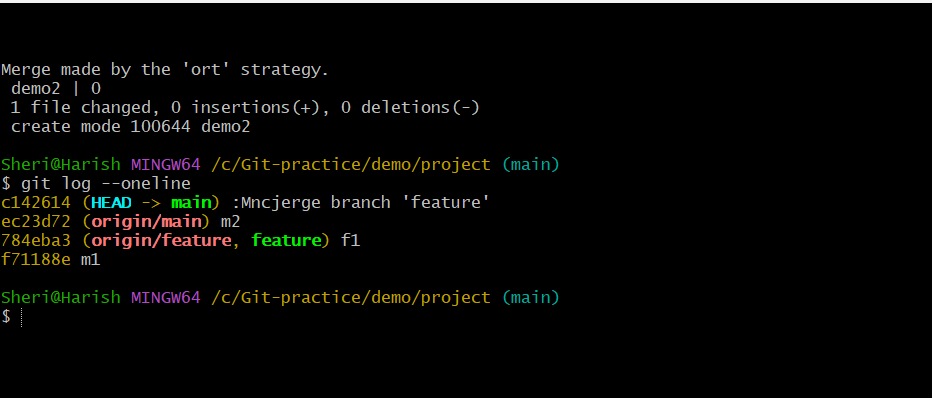


Fig: Merging of feature branch to the main branch using “**default merge**”

git log --oneline

c142614 (**HEAD** -> **main**) :Mncjerge branch 'feature'

ec23d72 (**origin/main**) m2

784eba3 (**origin/feature**, **feature**) f1

f71188e m1

Default merge, it Preserves full history with all commits and provides these commit history in **nonlinear manner** as show in above figure.

1. **Squash merge:**

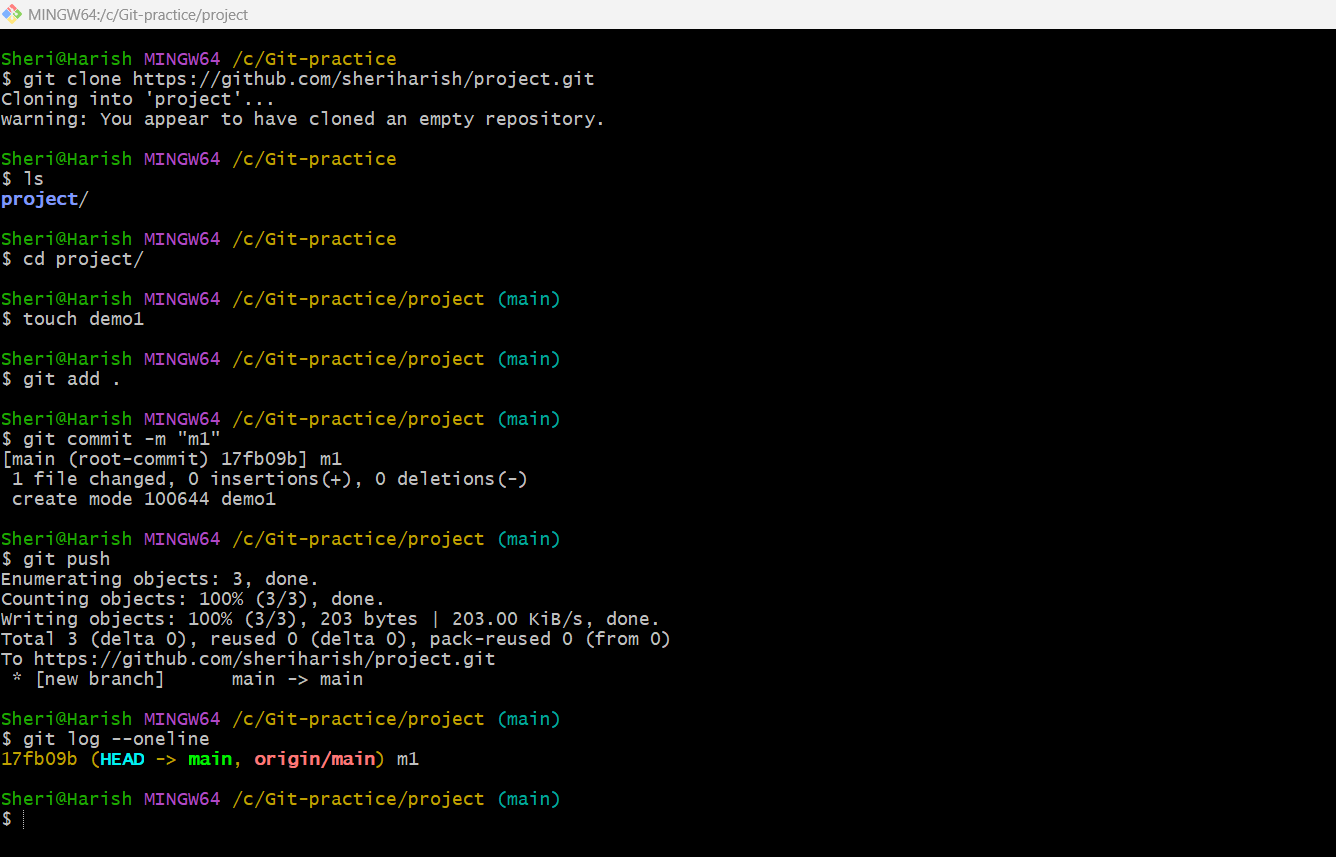
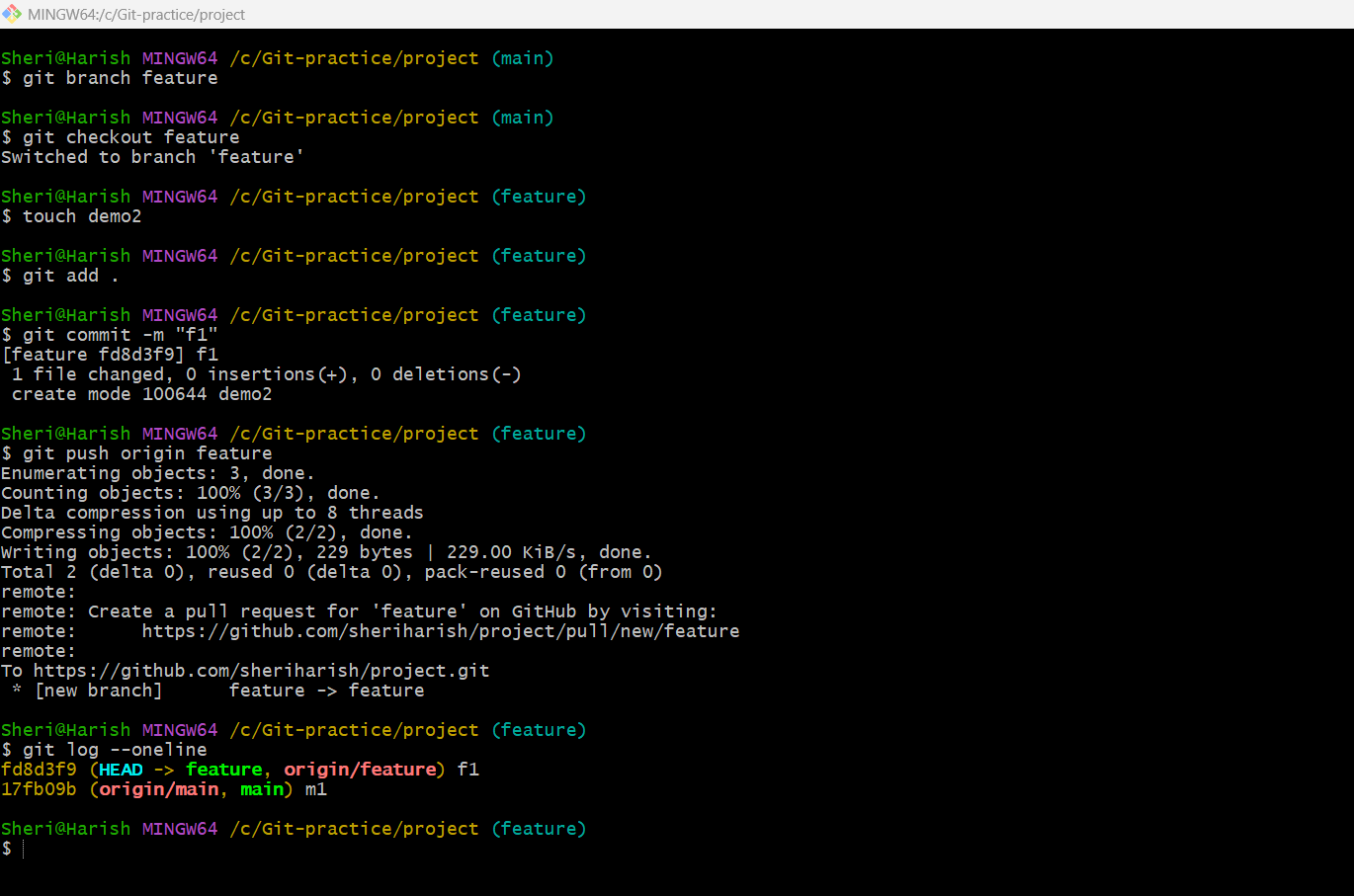
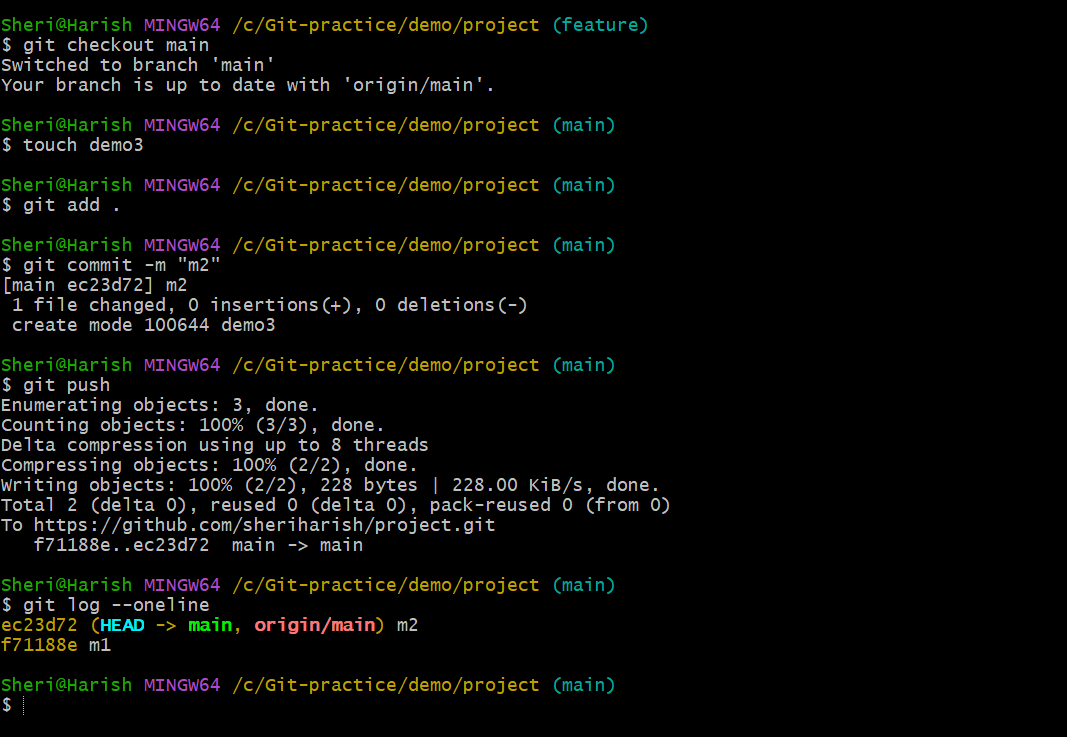
**Step1**: Create a commit “m1” by creating file in main branch (do it in new directory or folder).

Fig: In main branch ‘m1’ commit is done by create a file demo1.

**Step2:** Create a new branch **feature** and create a file (demo2) and commit it (f1).



**Step3:** By switching to the main branch again create a file and do commit as “m2”.



Step4: Now do the Squash merge from the main branch.

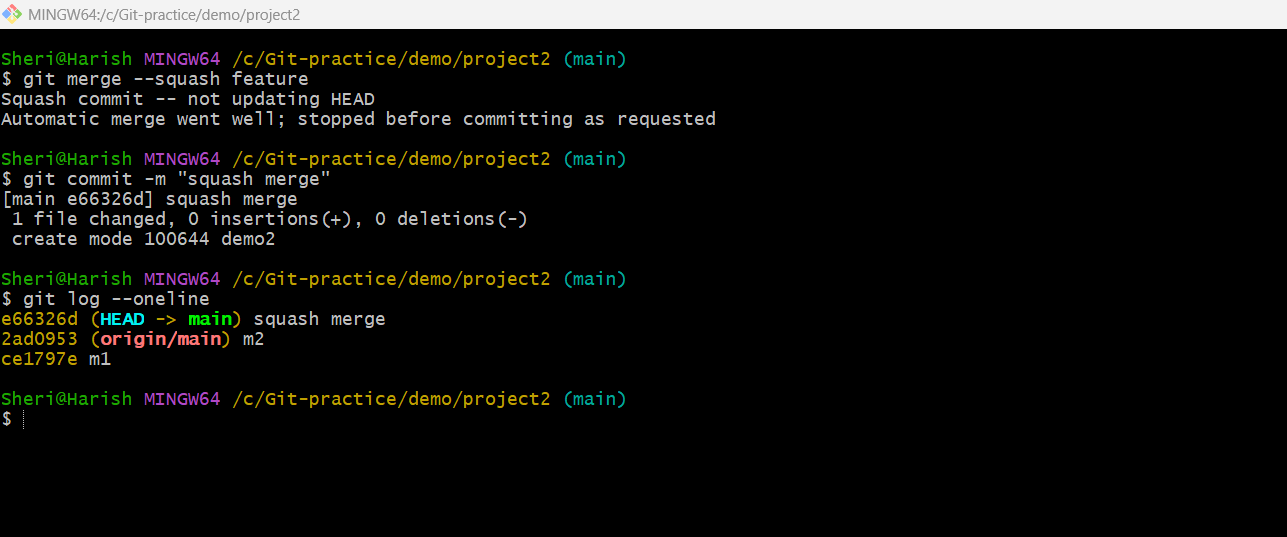


Fig: Squash merge.

**Note:** In squash merge whatever commits are done in feature branch are combined into a **single commit history** with commit name **“squash merge”.** So as a result **f1** commit is under the history of squash merge commit name.

In Simple words squash merge combines all commits into a single commit.

**Example:** If we do f1, f2 and f3 commits in feature branch, while performing the Squash merge these all commits f1, f2 and f3 are combines into single commit with the name of squash commit.

1. **Rebase merge:**

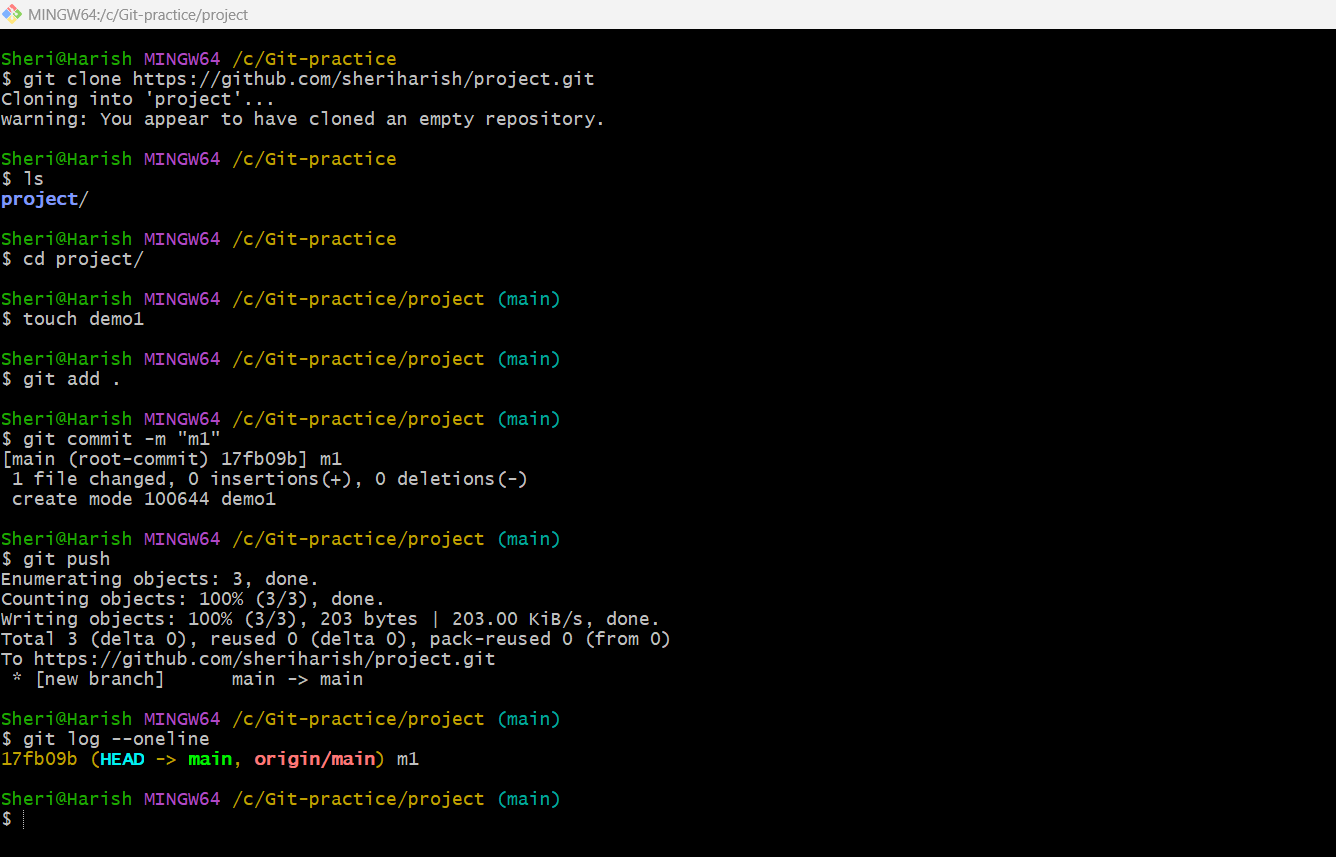
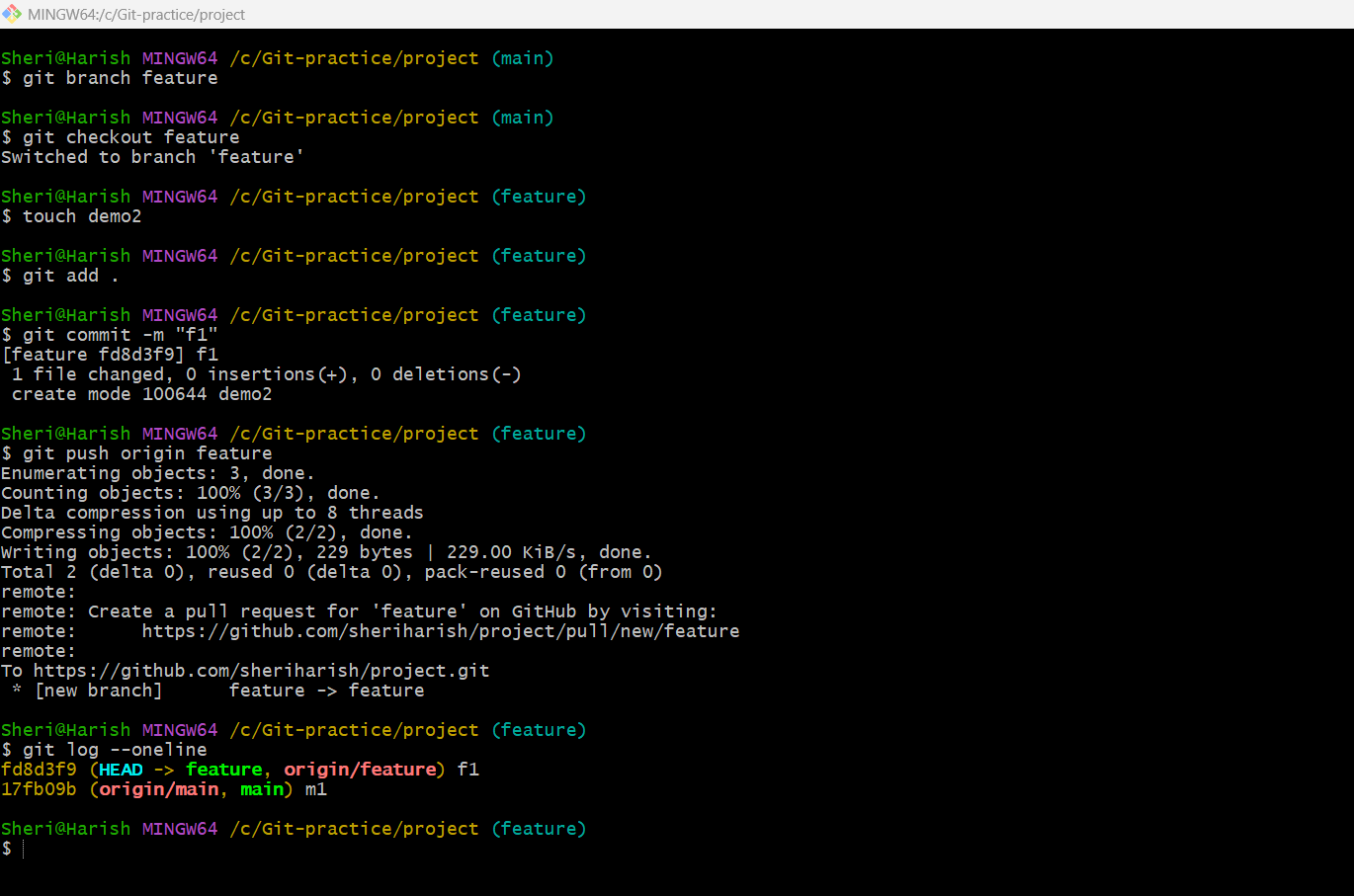
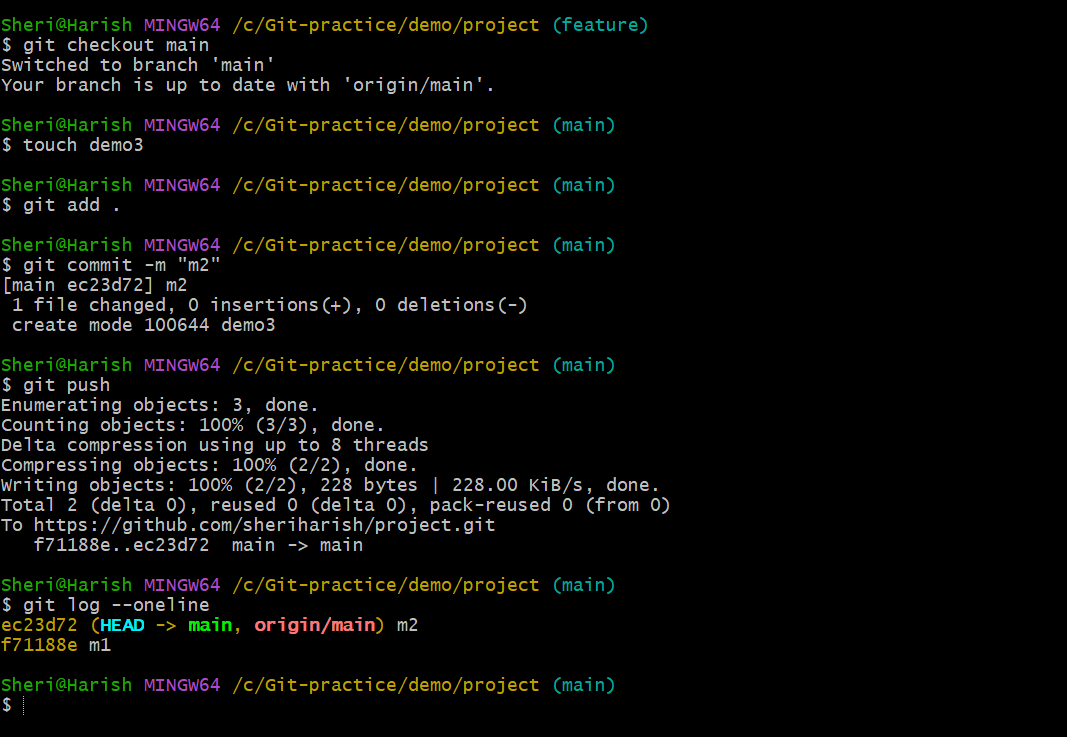
**Step1**: Create a commit “m1” by creating file in main branch (do it in new directory or folder).

Fig: In main branch ‘m1’ commit is done by create a file demo1.

**Step2:** Create a new branch **feature** and create a file (demo2) and commit it (f1).



**Step3:** By switching to the main branch again create a file and do commit as “m2”.



**Step4:** Now do the rebase merge from the main branch as well as from feature branch.

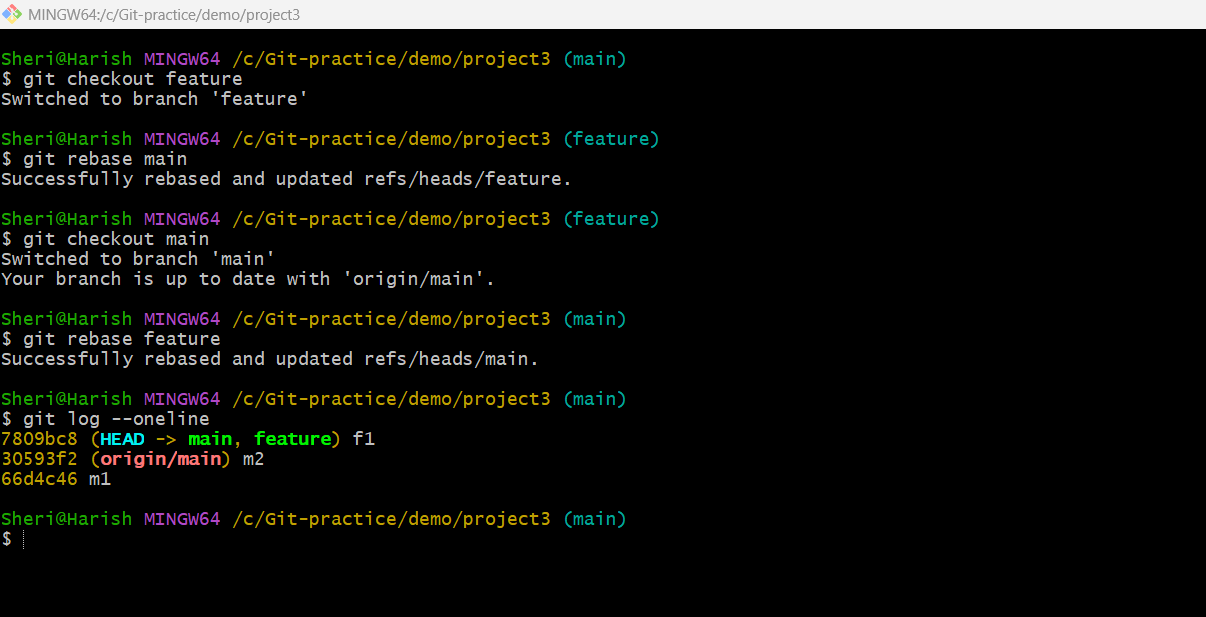


Fig: Rebase merge at both branches.

git log --oneline

7809bc8 (**HEAD** -> **main**, **feature**) f1

30593f2 (**origin/main**) m2

66d4c46 m1

**Note:** In rebase merge the commit history is provided in a linear way as shown in above figure.

**Difference between Default Merge, Squash Merge, and Rebase Merge in GitHub**

| **Merge Type** | **Default Merge (Merge Commit)** | **Squash Merge** | **Rebase Merge** |
| --- | --- | --- | --- |
| **Command** | git merge --no-ff feature-branch | git merge --squash feature-branch | git rebase main |
| **Commit History** | Preserves full history with all commits | Squashes multiple commits into a single commit | Creates a linear history by moving commits |
| **Merge Commit** | Yes (creates a new merge commit) | No (single squashed commit) | No (commits are rewritten) |
| **Commit Structure** | Retains all individual commits | Combines all commits into one | Reapplies commits on top of the target branch |
| **Best for** | Keeping track of branch history | Keeping history clean and concise | Making history linear without merge commits |
| **Pros** | - Keeps detailed commit history  - Good for tracking feature branches  - Useful for large teams | - Keeps main branch clean  - Avoids clutter from multiple small commits | - Creates a cleaner, linear history  - Makes bisecting easier for debugging |
| **Cons** | - Can clutter history with too many merge commits | - Loses individual commit details  - Harder to track who contributed what | - Can cause conflicts if the branch is outdated  - History rewriting can be dangerous |

**4. Cherry-pick**

It is used to merge/combined a specific commit from one branch (feature-branch) to another branch (main/master branch) without merging an entire branch.

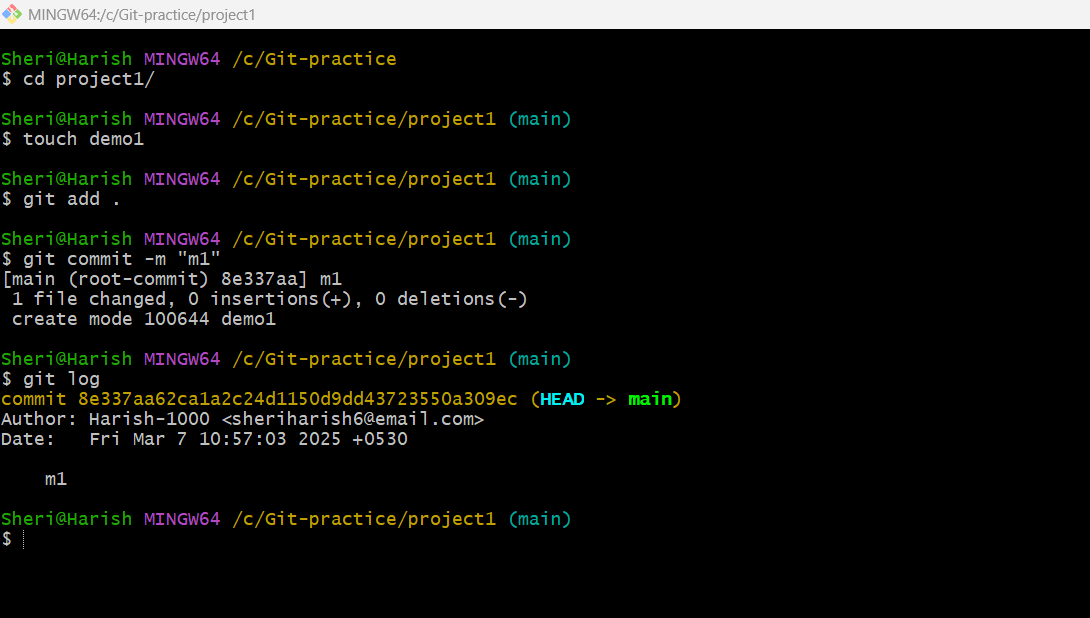
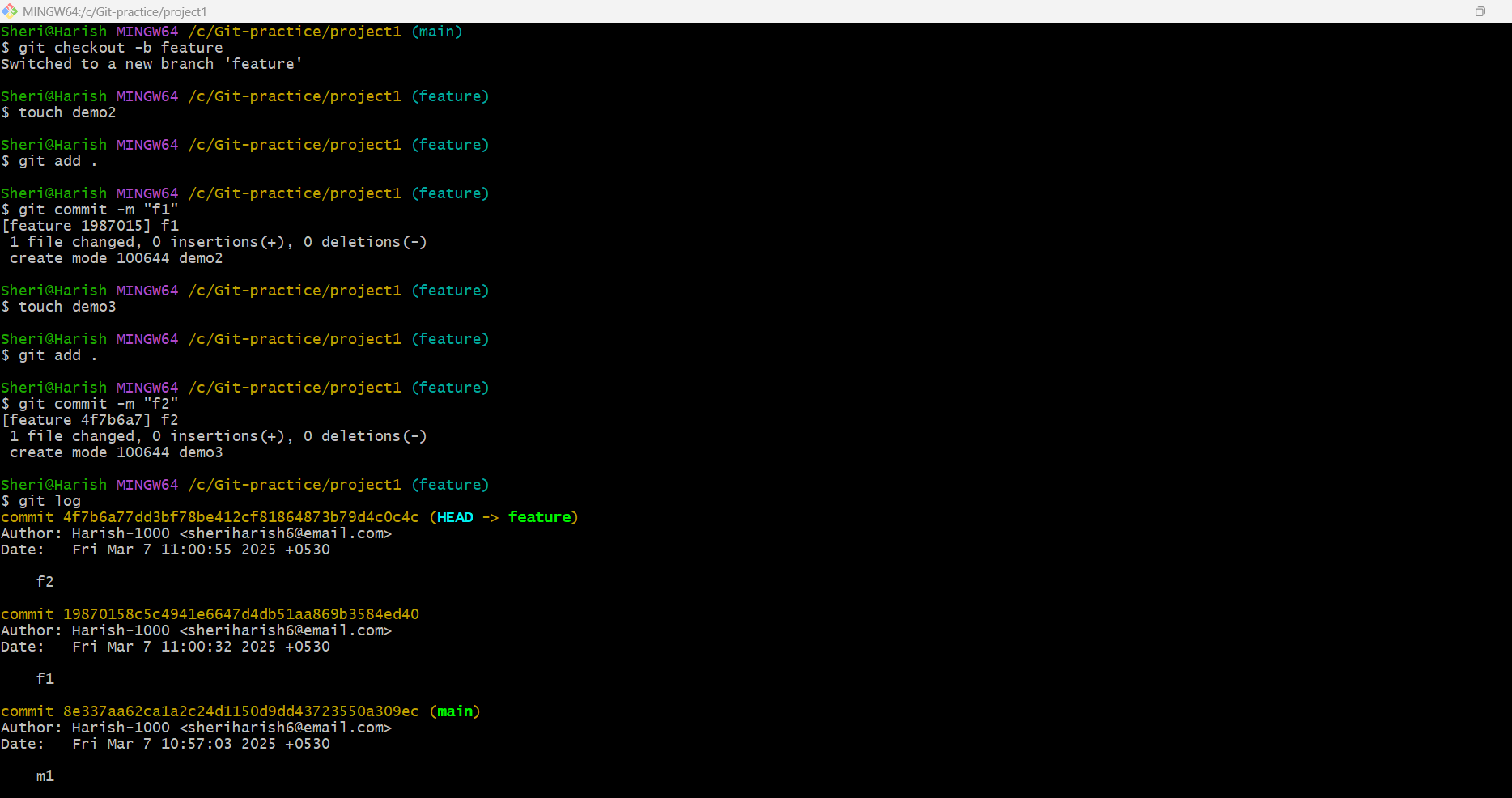
Step1: Create remote repository and in main branch perform a single commit “m1”.

Fig: Single commit “m1” in main branch.

**Step2:** Create another branch (feature) and perform two commits in it (f1 & f1).



In above figure the feature branch consist of three commits m1, f1, and f2.

**Step3:** Now do the **git cherry-pick** <commit ID> command to move “**f1**” commit of feature branch to the main branch.

**Note:** It is done by standing on the main branch.

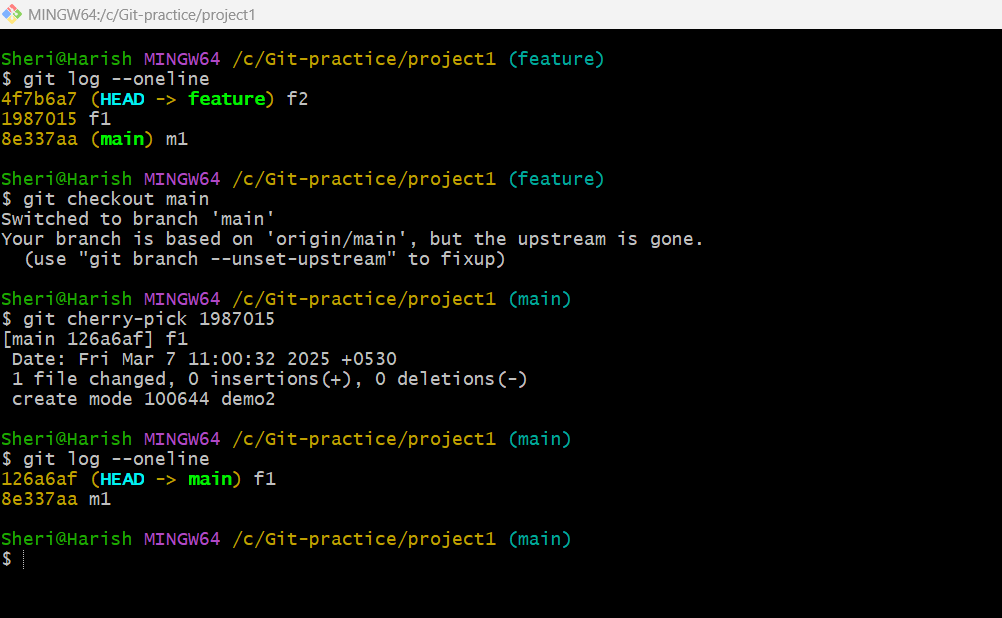


Fig: Commit “f1” is moved to the main branch using cherry-pick command.