**Undo concept in GitHub**

In Git and GitHub, **undoing changes** is a common task when you need to revert mistakes, discard unwanted commits, or restore previous states of your repository. There are several ways to undo changes depending on the situation. Below are the key **undo concepts** in Git and GitHub:

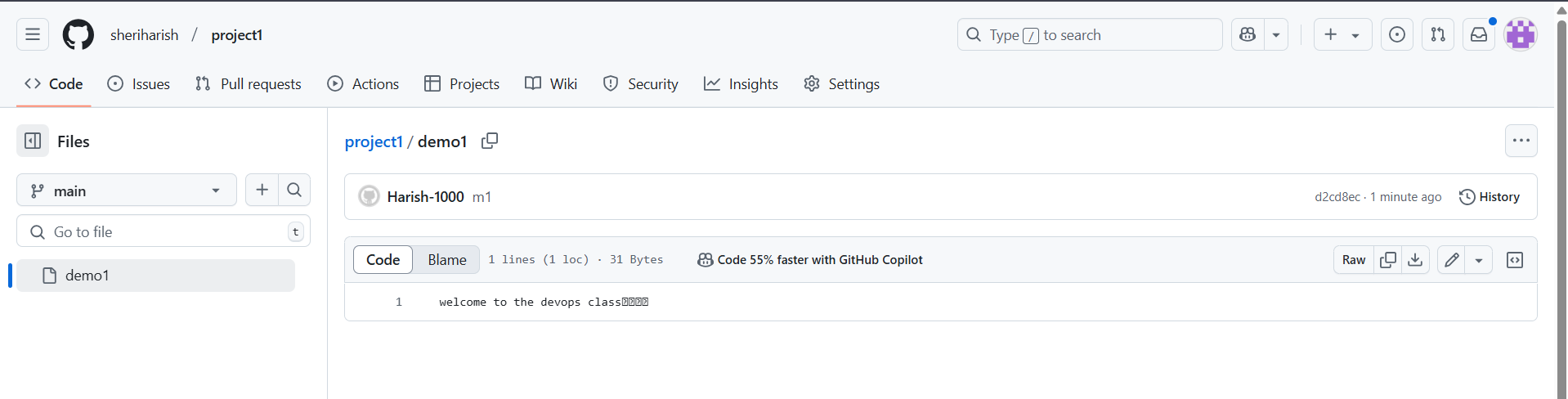
**1. Undo Uncommitted Changes**

If you have made changes to your working directory or staging area but haven't committed them yet, you can undo them by using

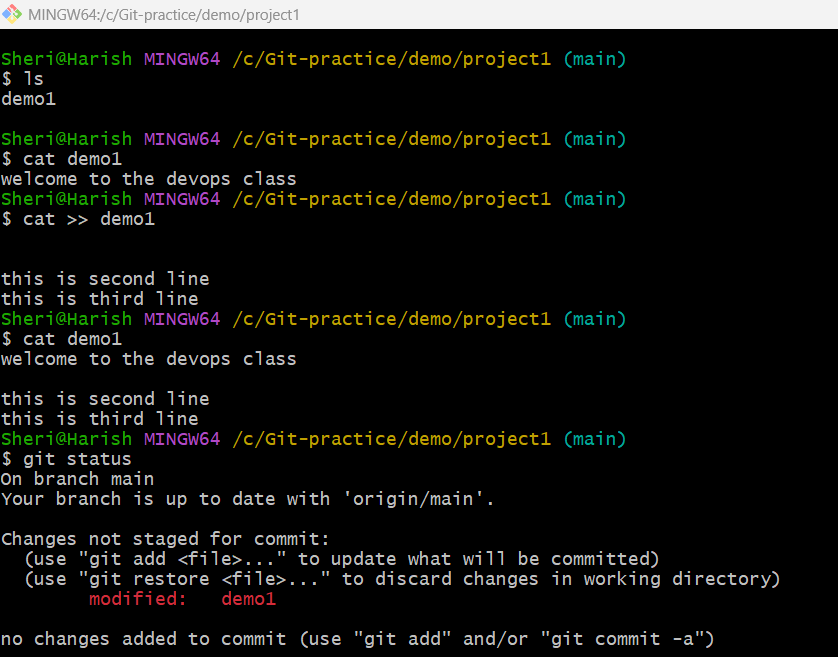
1. git checkout
2. git restore

**Perform it Practical:**

**Step1**: Create a repository and within it create a file with some content.

****

**Step2:** Write the some more lines of content in the file “demo1”. But don’t do “**git add”** and “**git commit”.** (It should be in working area then only we can do undo with checkout command).



**Step3:** Now do the “**git checkout”** or **git restore** to make undo of a file at working area.

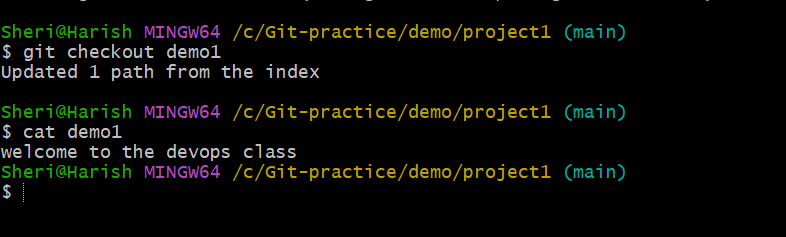


Fig: Doing undo with git checkout <file-name> command.

In above figure the newly added lines in the demo1 file are made undo. After doing git checkout only one existing (welcome to the devops class) line is remained.

Similarly making undo using git restore <file-name> command:

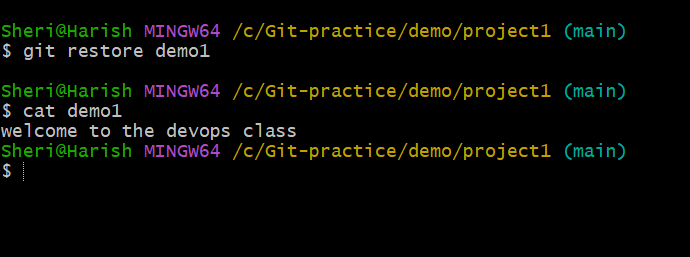


Fig: Doing undo Using git restore <file-name> command.

### 2. ****Undo Commits****

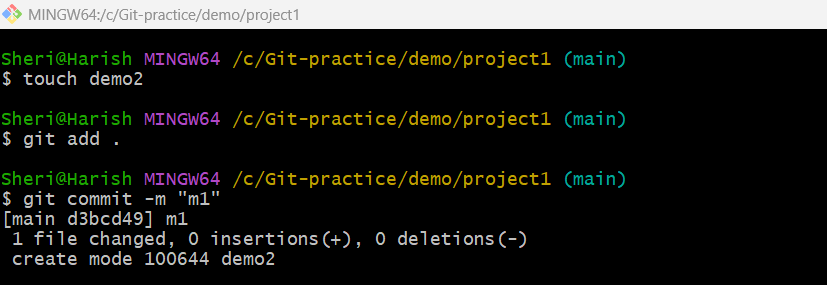
If you have already committed changes but want to undo them, there are several approaches.

1. Soft reset (git reset --soft HEAD~1)
2. Mixed reset (git reset --mixed HEAD~1)
3. Hard rest (git reset --hard HEAD~1)
4. Amend (git commit –amend –m <new-commit-name>)

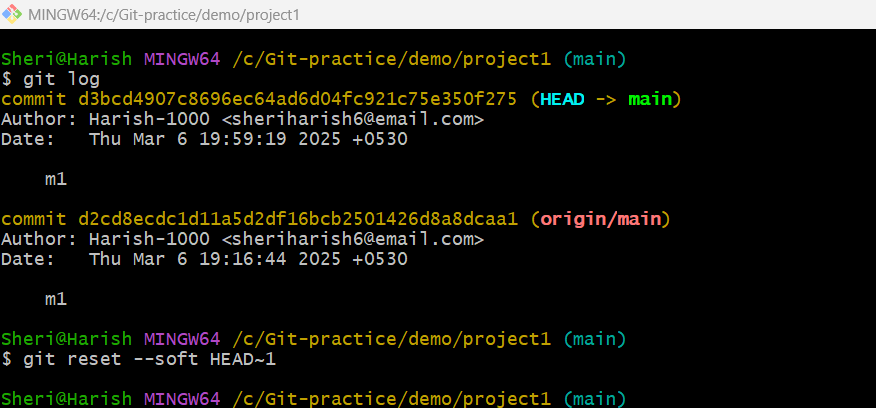
**Perform it practically:**

1. **Soft reset:** It is used to undo the changes from commit state to staging state.

Step1: Create a repository, add a file and commit it.



**Step2**: Now do the git reset –soft command to make undo the committed changes.



**Note:** When we do **git reset –soft HEAD~1** the file revert back to “index/staging area” from commit area”.As show in below figure.

Staging/index area

Commit area

Remote repository

git reset –soft HEAD~1

Working area

Fig: Soft reset.

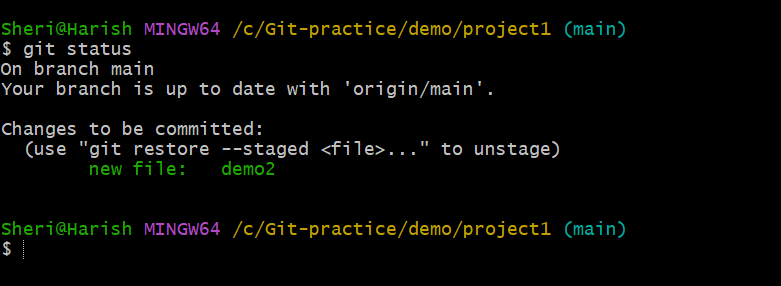
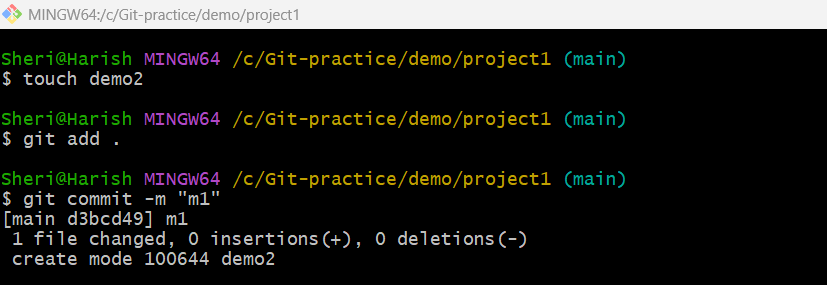


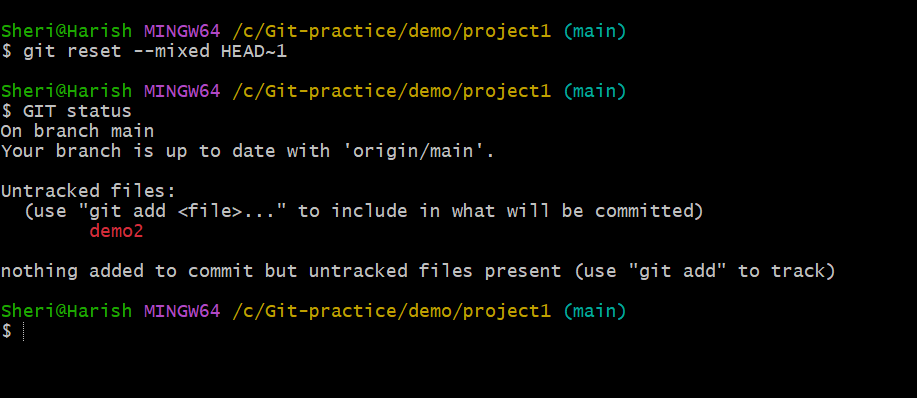
Fig: Demo2 file is at staging/index area.

1. **Mixed reset:** It is used to undo the changes from commit state to working state.

Step1: Create repository, create a file and make commit.



Step2: Now do git reset –mixed HEAD~1 command.



**Note:** When do the **git reset –mixed HEAD~1** command the demo2 file is rever back to the working area from the commit area. As show in above figure.

Remote repository

Working area

Staging/index area

Commit area

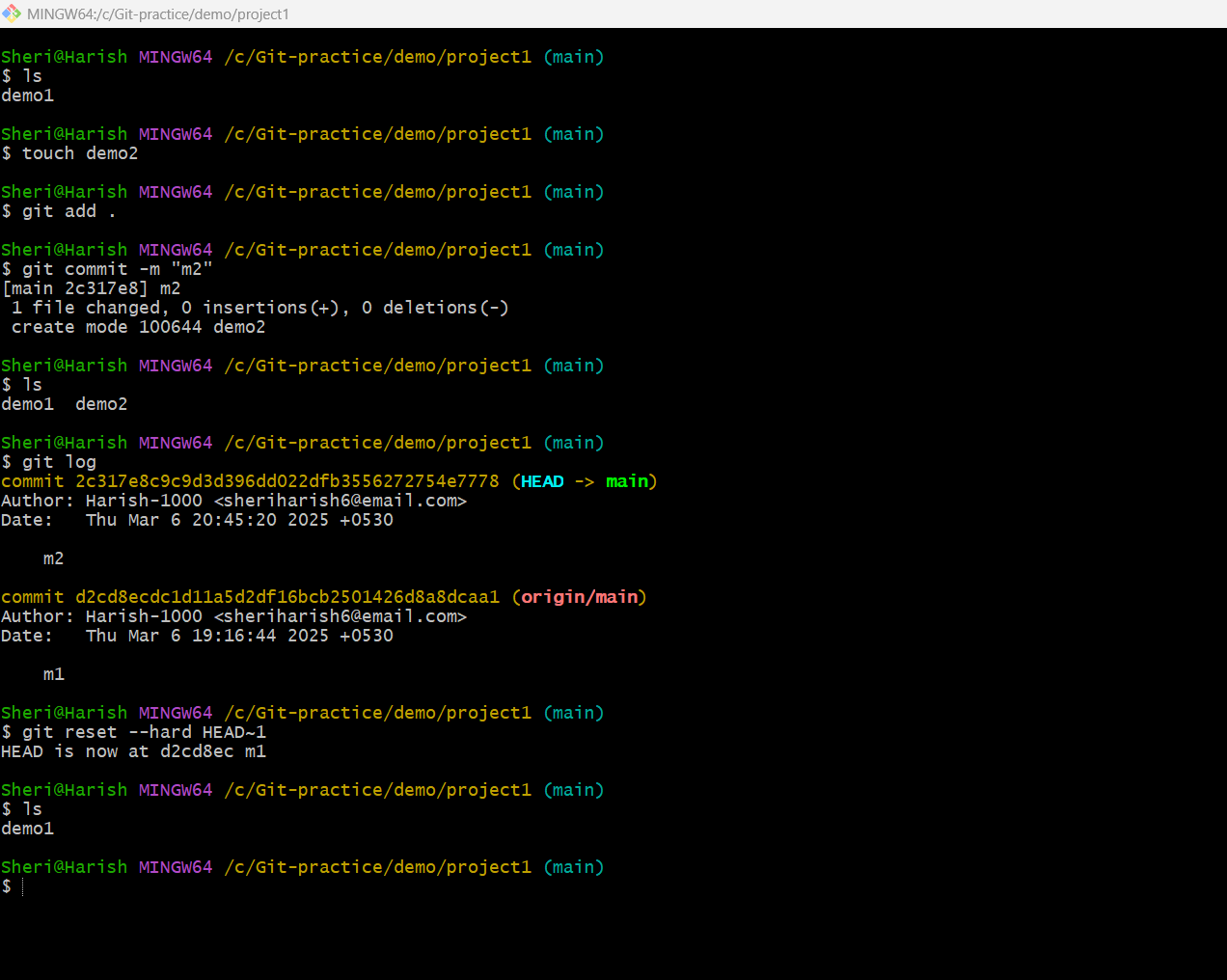
git reset –mixed HEAD~1

Fig: Mixed reset.

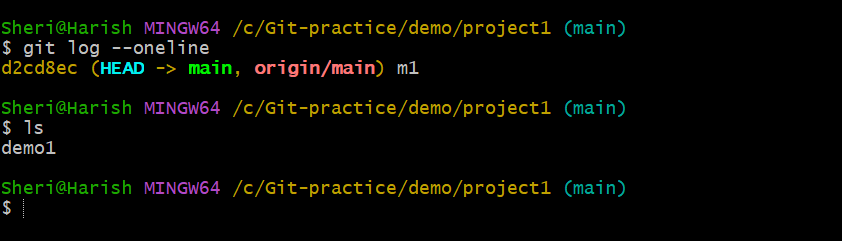
1. **Hard reset:**

It does **three things**:

1. Moves the current branch pointer (HEAD) to the specified commit.
2. Clears the staging area (index).
3. Deletes **all uncommitted changes** in the working directory.



In above figure last commit m2 and demo2 file is completely deleted.



**Note:** This **deletes** the last commit and all changes permanently. That means hard rest command deletes the contents from all stages like index, working and commit area.

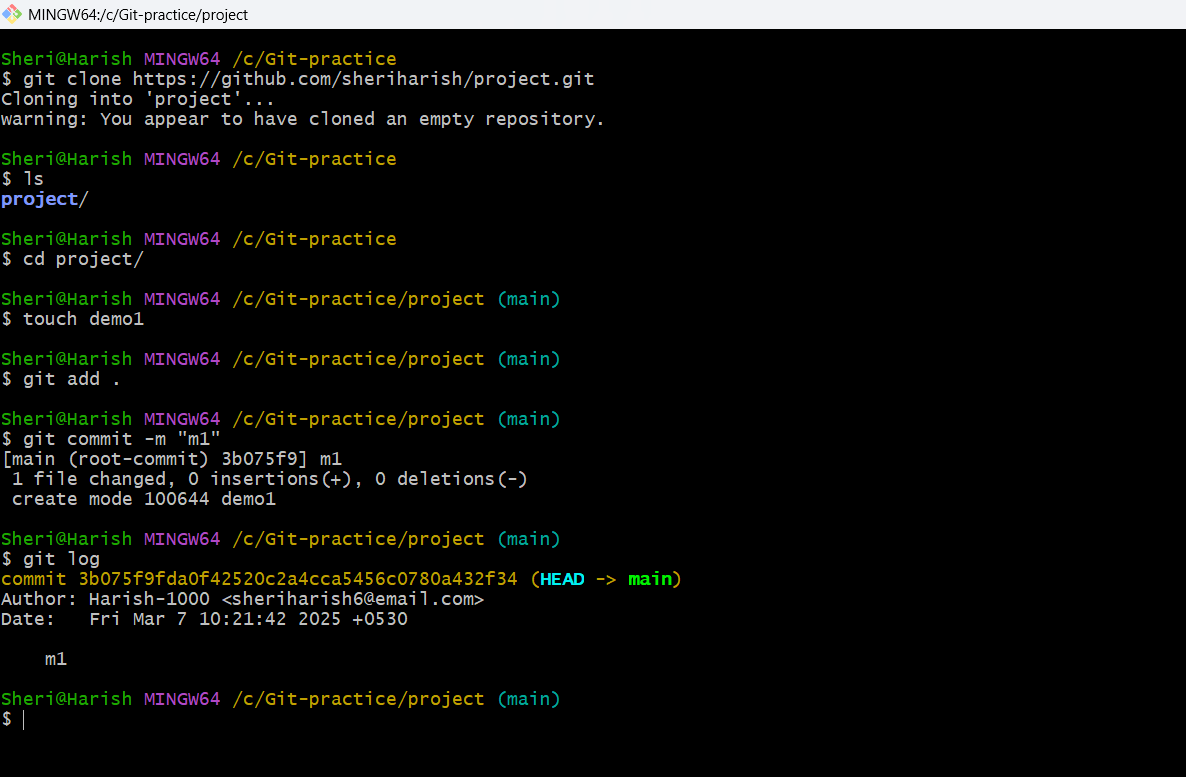
No history of the last commit is present in the local repository.

1. **amend:**

The **git commit --amend** command allows you to modify the most recent commit in your Git history. It is commonly used to fix commit messages/name, add missing changes, or update the commit without creating a new one.

**Note**: Mostly it is used to rename the commit, without moving to the stating/index area.

**Step1:** Create a repository and clone it to local machine and create a file (demo1) and commit it with a name **“m1”.**



**Step2:** Now do the **git commit –amend –m “<new-commit-name>”.** To change the commit name/message.

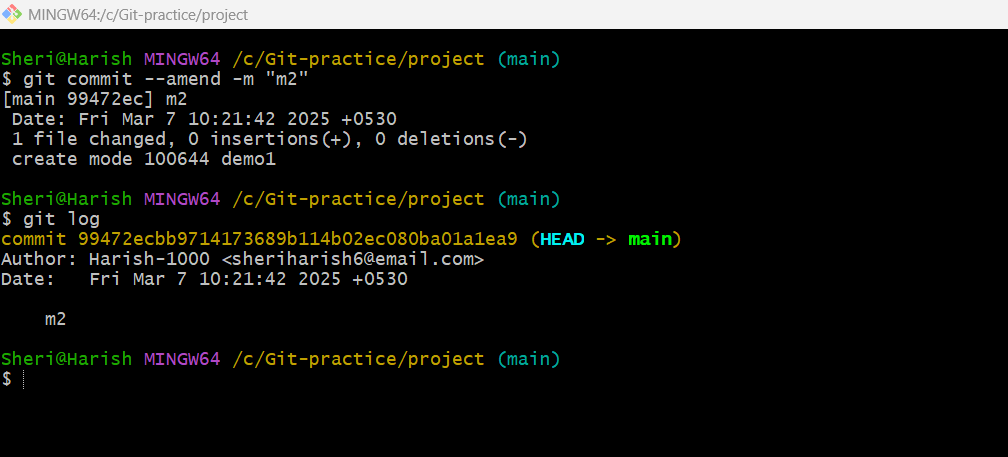


Fig: Commit name/message is changed to **m2**.

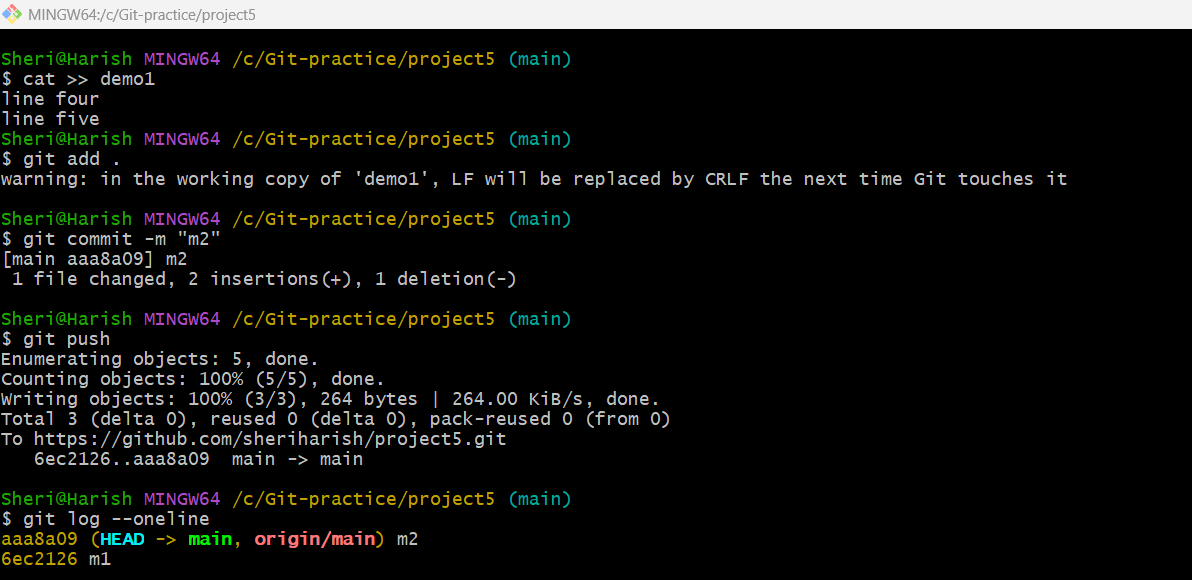
**Note:** While performing amend command it not only changes the commit name/message but also changes the commit ID.

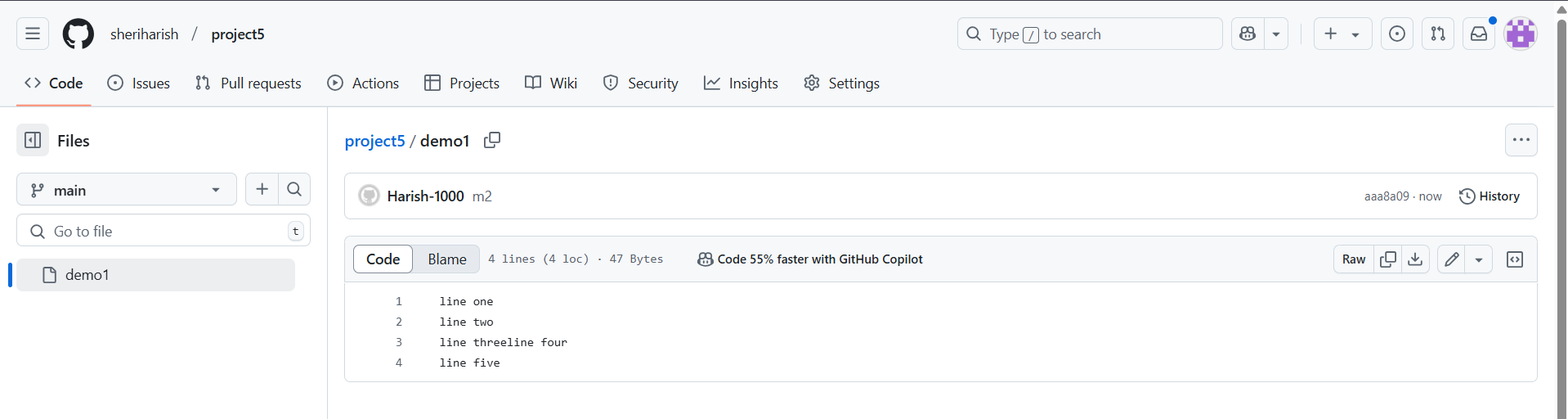
### 4. ****Undo a Push****

If you have pushed changes to a remote repository (e.g., GitHub) and want to undo them, you can use the following methods.

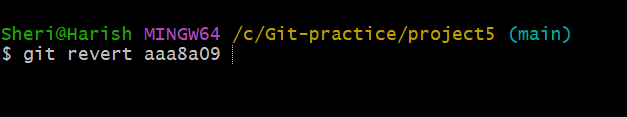
1. **revert.**

**Command:** git revert <commit-hash/ID> Ex: git revert d123456

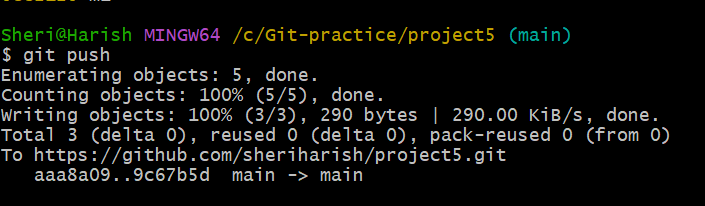


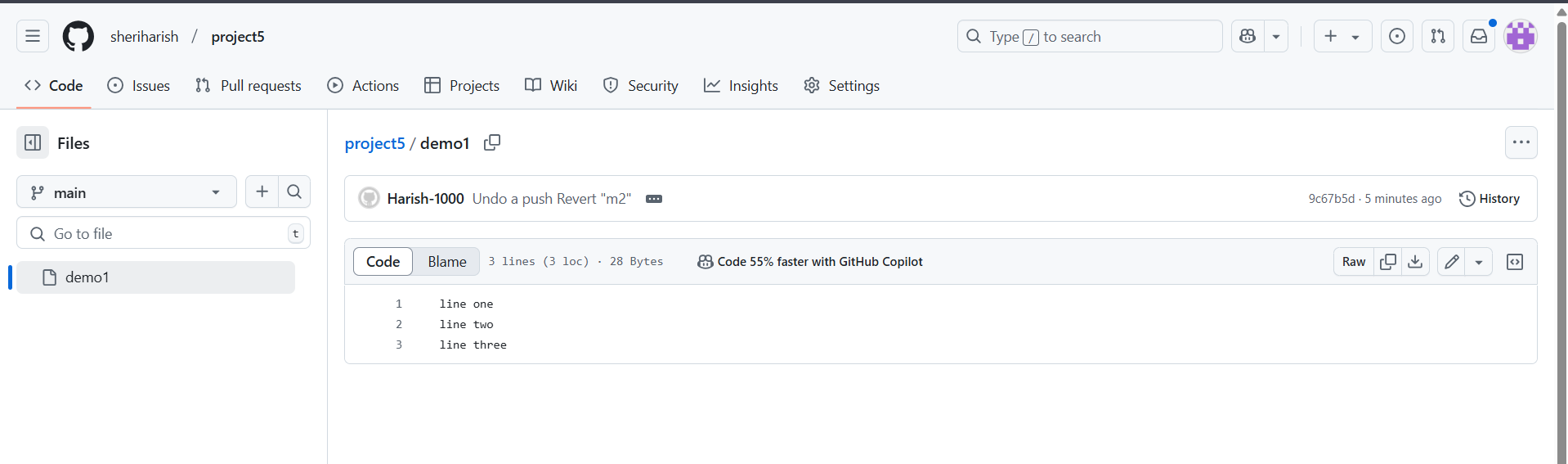


Now do the revert command to make undo a push.



**Note:** After doing revert we have to push again. Then only the undo will applied in remote repository.





**5. git stash (Pause command):**

* The **git stash** command in Git is used to temporarily save changes (pause the changes) in your working directory (working area) and staging area that are not ready to be committed.
* This allows you to switch branches, pull updates, or perform other tasks without losing your uncommitted work. The stashed changes (paused changes) can be reapplied later when you're ready to continue working on them.
* In simple words git stash command is used to temporarily pause or save changes in your working directory and staging area (index). This allows you to switch to another branch, commit important changes, and then return to your paused state to resume your work later.

Step1: Create a repository and perform a commit ‘m1” in main/master branch.

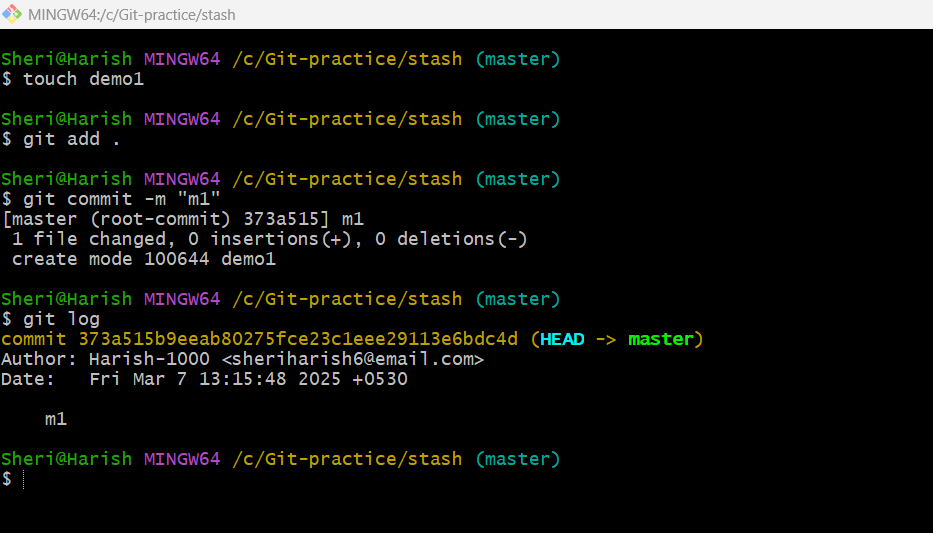
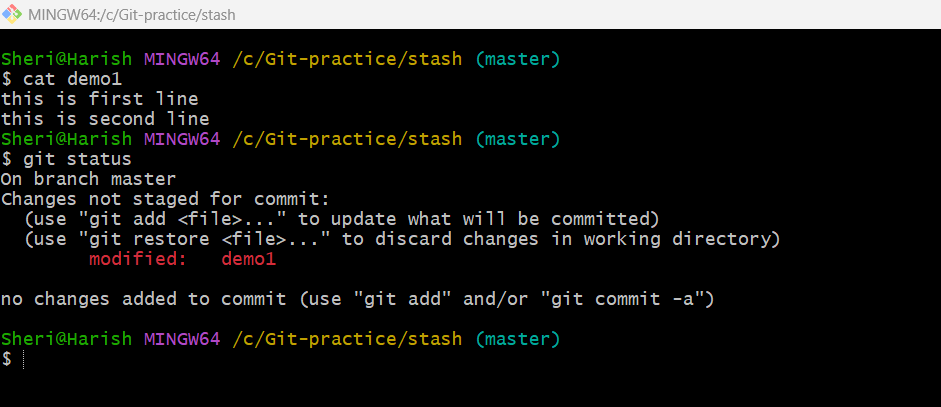


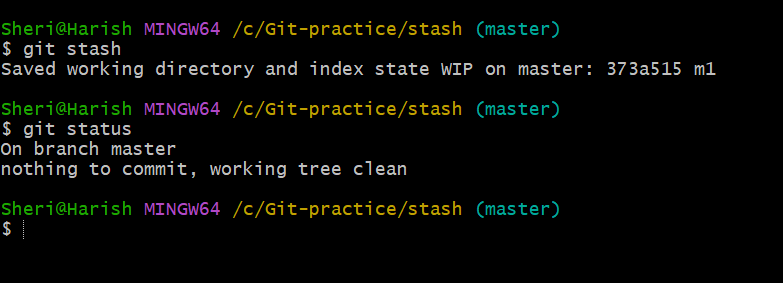
Fig: Commit “m1” in master branch.

Step2: Now write a content in the demo1 file.



Step3: Then do the **git stash** command.

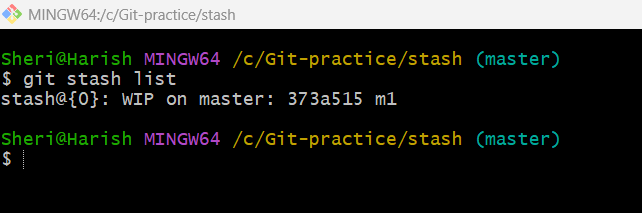
**#git stash:**



**Fig:** git stash.

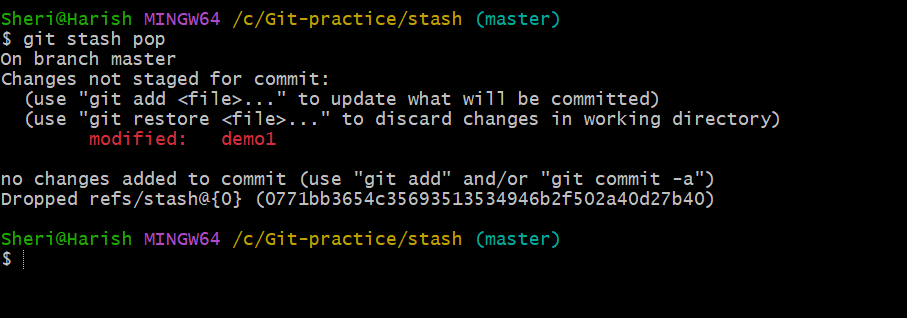
The demo1 file is saved/paused at working area, now you can switch branches, pull updates, or perform other tasks without losing your uncommitted work at working area of master branch.

#**git stash list:** The git stash list command displays a list of all stashed changes in your repository. Each stash entry is identified by an index (e.g., stash@{0}, stash@{1}, etc.)

****

**#git stash pop:** This command **applies the most recent stash to your working directory and removes/un-pause it from the stash list.**

**In simple word it unpassed the recent stash from stashing list.**



**#git stash drop:** This command is used to **delete a specific stash entry** from the stash list without applying it.

