VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Machhe, Belagavi, Karnataka-590018

Lab Experiment Record

Project Management with Git [BCSL358C]

Submitted in partial fulfillment towards AEC of 3rd semester of

Bachelor of Engineering
in
Computer Science and Engineering
(Artificial Intelligence & Machine Learning)

Submitted by

MEGHANA M
4GW24CI025

DEPARTMENT OF CSE (Artificial Intelligence & Machine Learning)

GSSS INSTITUTE OF ENGINEERING & TECHNOLOGY FOR WOMEN

(Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi & Govt. of Karnataka)

K.R.S ROAD, METAGALLI, MYSURU-570016, KARNATAKA

(Accredited by NAAC)

2025-2026

Table of Contents

Sl.No	Experiments	Page.No
1.	**Environmental Preparation**	2
2.	**Setting Up and Basic** Commands Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.	3
3.	**Creating and Managing Branches** Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."	4
4.	**Creating and Managing Branches** Write the commands to stash your changes, switch branches, and then apply the stashed changes	5
5.	**Collaboration and Remote Repositories** Clone a remote Git repository to your local machine.	6
6.	**Collaboration and Remote Repositories** Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.	7
7.	**Collaboration and Remote Repositories** Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.	8
8.	**Git Tags and Releases** Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.	9
9.	**Advanced Git Operations** Write the command to cherry-pick a range of commits from "sourcebranch" to the current branches	10-12
10.	**Analysing and Changing Git History** Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?	13
11.	**Analysing and Changing Git History** Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12- 31."	14
12.	**Analysing and Changing Git History** Write the command to display the last five commits in the repository's history.	15
13.	**Analysing and Changing Git History** Write the command to undo the changes introduced by the commit with the ID "abc123".	16

Appendix: https://github.com/meghanamurali10-crypto/4GW24CI025.git

1. Environmental Preparation (Before Experiment 1)

The following steps are required to reach the starting point of **Experiment 1**:

- **Create a Project Directory:** Before we can "initialize a repository in a directory," we must create that directory on your system.
  - **Command:** `mkdir my_project`
  - **Navigate to the Directory:** We must be inside the folder where we want our Git history to live.
  - **Command:** `cd my_project`

2. Initial Identity Configuration

To complete the "commit" requirement of Experiment 1, **Git needs to know who is making the changes**. This is typically done through configuration commands which are not detailed in the syllabus but are a prerequisite for the tool to function correctly:

- **Set Username:** `git config --global user.name "Your Name"`
- **Set Email:** `git config --global user.email "yourname@example.com"`
- **Usage:** These commands are only run once on a new system to ensure that our commit history in Experiment 1 is properly attributed.

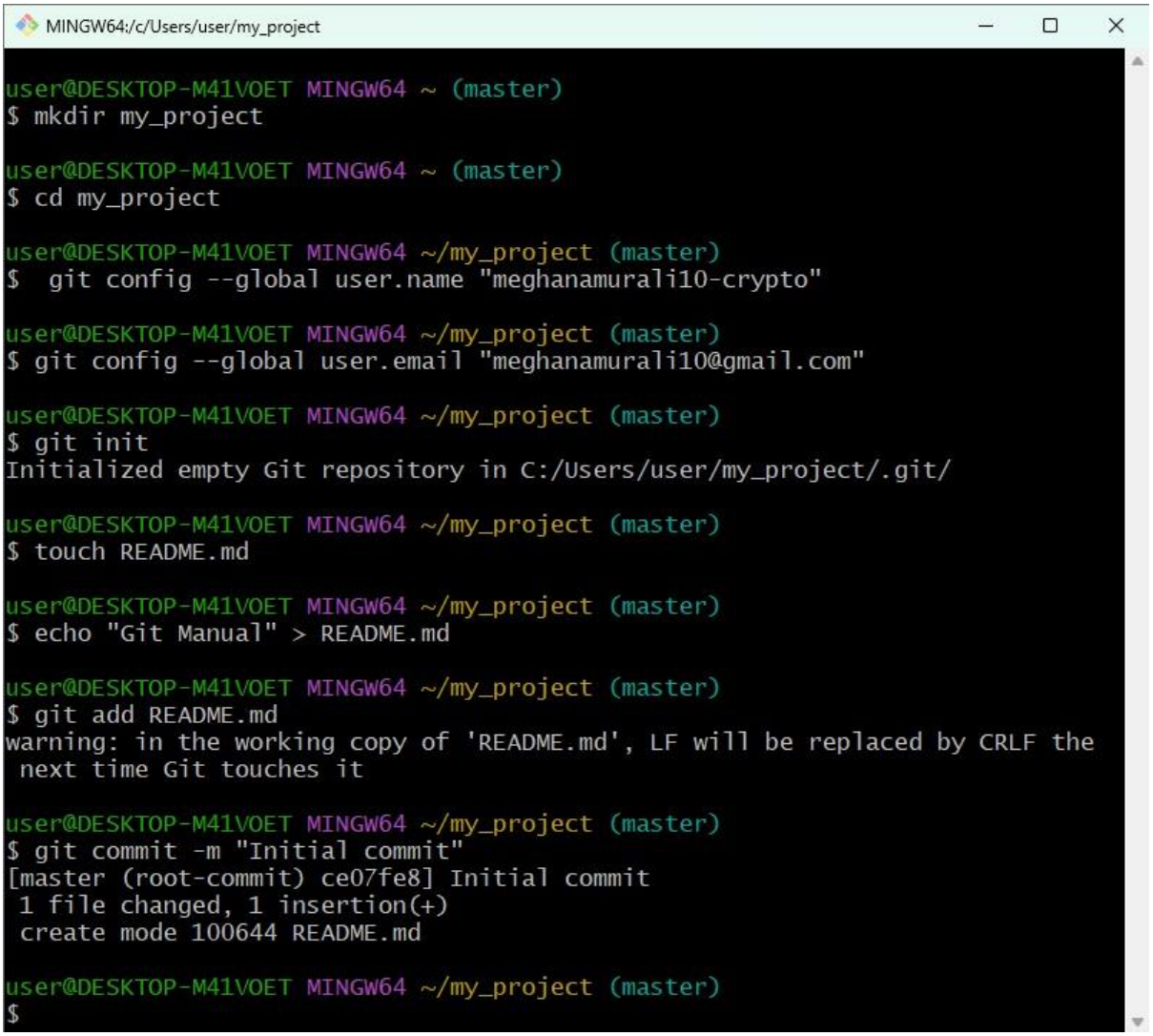
3. The First Syllabus Command: Starting Experiment 1

- **The Command:** `git init`
- **Usage:** This is the command used to **"Initialize a new Git repository in a directory"**.
- **What it does:** It creates a hidden folder (named `.git`) that begins tracking every change we make from that moment forward.

Analogy for `git init`: Starting Experiment 1 with `git init` is like **opening a new notebook and writing your name on the cover**. Before we can write our first page (a commit), we must first designate that specific notebook as the place where our project's history will be recorded.

Experiment 1: Setting Up and Basic Commands

- **Goal:** Start a new project and save the first version.
- **Command 1:** `git init` — **Initialises** a new Git repository in the folder. It creates a hidden `.git` folder to track changes.
- **Command 2:** `git add <filename>` — Moves a file to the **Staging Area**. This is like "checking in" the items before they are permanently saved.
- **Command 3:** `git commit -m "Your Message"` — Permanently saves the staged changes to the project history with a descriptive label.
- **Note:** The **Staging Area** allows us to review exactly what we are about to save before we commit it permanently.



Experiment 2: Creating and Managing Branches

- **Goal:** Work on a new feature without breaking the main code.
- **Command 1:** `git branch feature-branch` — Creates a new separate line of development named "feature-branch".

  `git add README.md` → Stages changes made to the *README.md* file.

  `git commit -m "Branch update"` → Commits the staged changes with a descriptive message

- **Command 2:** `git checkout master` — **Switches** from working view back to the "master" (main) branch.
- **Command 3:** `git merge feature-branch` — Combines the work from the feature branch back into the main master branch.
- **Note:** A **branch** is like a parallel universe where we can experiment safely; **merging** brings those experiments into reality.

```
user@DESKTOP-M41VOET MINGW64 ~/my_project (master)
$ git branch feature-branch

user@DESKTOP-M41VOET MINGW64 ~/my_project (master)
$ git checkout feature-branch
Switched to branch 'feature-branch'

user@DESKTOP-M41VOET MINGW64 ~/my_project (feature-branch)
$ git add README.md

user@DESKTOP-M41VOET MINGW64 ~/my_project (feature-branch)
$ git commit -m "Branch update"
On branch feature-branch
nothing to commit, working tree clean

user@DESKTOP-M41VOET MINGW64 ~/my_project (feature-branch)
$ git checkout master
Switched to branch 'master'

user@DESKTOP-M41VOET MINGW64 ~/my_project (master)
$ git merge feature-branch
Already up to date.
```

Experiment 3: Creating and Managing Branches - Stashing Changes

- **Goal:** Temporarily hide our work to do something else.
- **Command 1:** `git stash` — Temporarily "shelves" the uncommitted changes so we have a clean folder.
- **Command 2:** `git stash apply` — Brings our shelved changes back to the surface so we can continue working.

Stash Uncommitted Changes

- **Goal:** Save the current uncommitted changes (modifications to tracked files) without committing them.
- **Command:** `git stash push -m "WIP: Descriptive feature name"`

Switch to a New Branch

- **Goal:** Create a new feature branch and switch the working directory to it.
- **Command:** `git checkout -b <new-branch-name>`

Apply the Stashed Changes

- **Goal:** Retrieve the saved work-in-progress and apply it to the new branch.
- **Command (Recommended):** `git stash pop`
- **Note: Stashing** is essential when we are interrupted mid-task and need to switch branches quickly without losing progress.

```
user@DESKTOP-M41VOET MINGW64 ~/my_project (master)
$ git stash
No local changes to save

user@DESKTOP-M41VOET MINGW64 ~/my_project (master)
$ git stash apply
No stash entries found.
```

```
user@DESKTOP-M41VOET MINGW64 ~/my_project (master)
$ git stash push -m "WIP on master: Feature A"
No local changes to save

user@DESKTOP-M41VOET MINGW64 ~/my_project (master)
$ git checkout -b feature/A-new-branch
Switched to a new branch 'feature/A-new-branch'

user@DESKTOP-M41VOET MINGW64 ~/my_project (feature/A-new-branch)
$ git stash apply
No stash entries found.

user@DESKTOP-M41VOET MINGW64 ~/my_project (feature/A-new-branch)
$ git stash pop
No stash entries found.
```

Experiment 4: Collaboration and Remote Repositories - Cloning Remote Repositories

- **Goal:** Get a copy of an existing project from a server.
- **Command:** `git clone <URL>` — Downloads a full copy of a project (including all its history) from a remote server (like GitHub) to our local computer.
- **Note: Cloning** is different from just downloading a ZIP file because it includes the entire Git history of the project.

```
user@DESKTOP-M41VOET MINGW64 ~/my_project (master)
$ git clone https://github.com/meghanamurali10-crypto/4GW24CI025.git
Cloning into '4GW24CI025'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 16 (delta 2), reused 6 (delta 1), pack-reused 0 (from 0)
Receiving objects: 100% (16/16), done.
Resolving deltas: 100% (2/2), done.
```

```
user@DESKTOP-M41VOET MINGW64 ~/my_project (master)
$ cd 4GW24CI025

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git remote -v
origin  https://github.com/meghanamurali10-crypto/4GW24CI025.git (fetch)
origin  https://github.com/meghanamurali10-crypto/4GW24CI025.git (push)
```

Experiment 5: Collaboration and Remote Repositories - Fetch and Rebase

- **Goal:** Update our local work with the latest changes from others.
- **Command 1:** `git fetch` — Downloads the latest updates from the remote server but **does not** merge them into our files yet.
- **Command 2:** `git rebase` — Moves our local changes so they appear **after** the latest updates you just fetched, keeping a straight project line.
- **Note: Rebasing** creates a cleaner, more linear project history compared to merging.

```
user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git fetch origin
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 967 bytes | 193.00 KiB/s, done.
From https://github.com/meghanamurali10-crypto/4GW24CI025
   3c64823..4a92531  main       -> origin/main

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git rebase
Successfully rebased and updated refs/heads/main.
```

Experiment 6: Collaboration and Remote Repositories - Custom Merge Messages

- **Goal:** Document why two branches are being combined.
- **Command:** `git merge feature-branch -m "Custom message here"` — Merges the branch and allows us to write a specific note about why this merge is happening.
- **Note:** Clear merge messages are vital in professional teams to understand why specific features were added at certain times.

Experiment 7: Git Tags and Releases

- **Goal:** Mark a specific point in history as a "Release".
- **Command:** `git tag v1.0` — Creates a **lightweight tag** named "v1.0" on your current commit.
- **Note: Tags** are like permanent bookmarks; they are usually used to mark stable software versions like v1.0, v2.0, etc.

```
user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git tag v1.0

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git log --oneline
4a92531 (HEAD -> main, tag: v1.0, origin/main, origin/HEAD, master, featur
e-branch) Add Git manual experiment 5 execution to README
3c64823 Add experiment note to remote_work.txt using GitHub
b1b66a2 Add file for Experiment 5 push
5ee548e New folder added
a4ba270 Create README.md
9f16891 Initial commit
```

```
user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git push origin v1.0
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/meghanamurali10-crypto/4GW24CI025.git
 * [new tag]         v1.0 -> v1.0
```

Experiment 8: Advanced Git Operations - Cherry-Picking

- **Goal:** Copy a specific change from one branch to another.
- **Command:** `git cherry-pick <commit_range>` — Selects specific commits from a "source-branch" and applies them to our current branch.
- **Note:** Use this if we only want one specific bug fix from another branch without taking all the other changes from that branch.

This procedure demonstrates how to selectively copy a specific commit (`Commit 2: Critical bug fix`) from the `feature-branch` onto the `main` branch, without including other commits from the source branch.

I. Setup: Create Branches and Commits

- `echo "Initial content." > file.txt` – Create a starting file.
- `git add file.txt` – Stage the file.
- `git commit -m "Initial commit"` – Create the base commit on `main`.
- `git checkout -b feature-branch` – Create and switch to the branch containing the fix.
- `echo "Feature-specific change." >> feature.txt` – Add a change that we **do not** want on `main` (Commit 1).
- `git add feature.txt`
- `git commit -m "Commit 1: New feature implementation"`
- `echo "BUG: Fixed critical error." >> file.txt` – Add the specific bug fix (Commit 2).
- `git add file.txt`
- `git commit -m "Commit 2: Critical bug fix for file.txt"` – **This is the commit to cherry-pick.**

II. Execution: Cherry-Pick the Commit

- `git log --oneline --skip 1 -n 1` – **Crucial Step:** Execute this to get the unique **Commit ID** (e.g., `c3b1a2d`) of the bug fix commit.
- `git checkout main` – Switch to the target branch where the fix is needed.
- `git cherry-pick <Commit_ID>` – Apply the exact changes from the copied Commit ID to `main` as a new commit.

III. Verification

- `git log --oneline` – Check the log on `main`. It should show the initial commit and the bug fix commit (Commit 2), but not the unwanted Commit 1.
- `cat file.txt` – Verify that the file now contains the "BUG: Fixed critical error" line.
- `ls feature.txt` – Verify this file **does not exist** on `main`, proving Commit 1 was skipped.

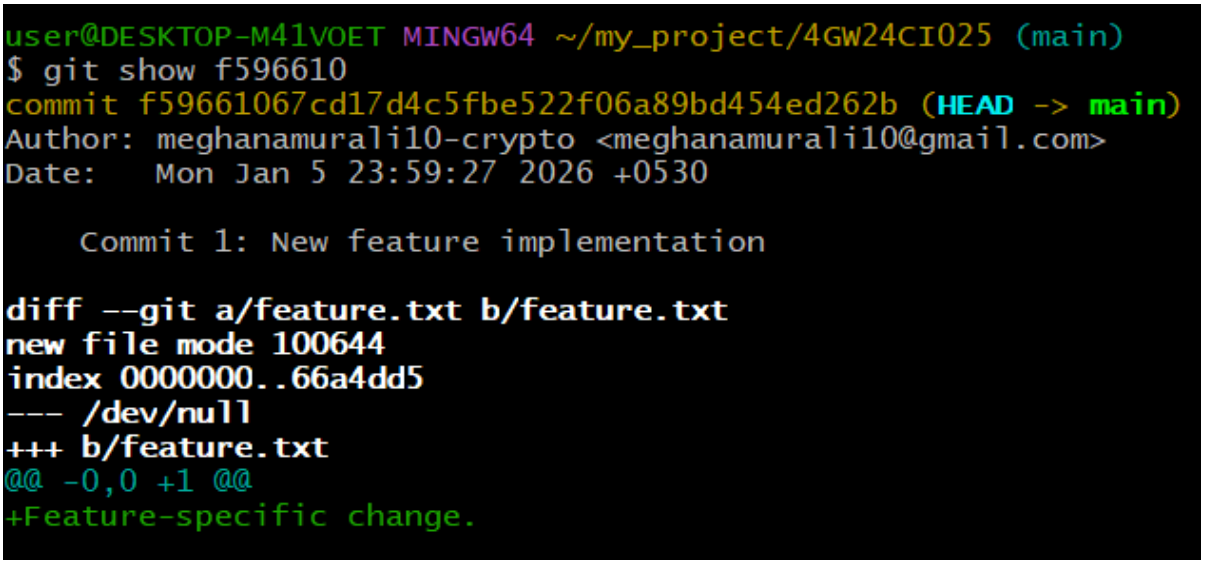
```
user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ echo "Initial content." > file.txt

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next tim
e Git touches it

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git commit -m "Initial commit"
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git checkout -b feature-branch
fatal: a branch named 'feature-branch' already exists

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git checkout feature-branch
Switched to branch 'feature-branch'

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (feature-branch)
$ echo "Feature-specific change." > feature.txt

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (feature-branch)
$ git add feature.txt
warning: in the working copy of 'feature.txt', LF will be replaced by CRLF the next
time Git touches it
```

```
user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (feature-branch)
$ git commit -m "Commit 1: New feature implementation"
On branch feature-branch
nothing to commit, working tree clean

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (feature-branch)
$ echo "BUG: Fixed critical error" >> file.txt

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (feature-branch)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next tim
e Git touches it

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (feature-branch)
$ git commit -m "Commit 2: Critical bug fix for file.txt"
[feature-branch 65a4eac] Commit 2: Critical bug fix for file.txt
 1 file changed, 1 insertion(+)

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (feature-branch)
$ git log --oneline --skip 1 -n 1
2c476a3 Commit 2: Critical bug fix for file.txt

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (feature-branch)
$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)
```

```
user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git cherry-pick 2c476a3
Auto-merging file.txt
CONFLICT (add/add): Merge conflict in file.txt
error: could not apply 2c476a3... Commit 2: Critical bug fix for file.
txt
hint: After resolving the conflicts, mark them with
hint: "git add/rm <pathspec>", then run
hint: "git cherry-pick --continue".
hint: You can instead skip this commit with "git cherry-pick --skip".
hint: To abort and get back to the state before "git cherry-pick",
hint: run "git cherry-pick --abort".
hint: Disable this message with "git config set advice.mergeConflict f
alse"

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main|CHERRY-PICK
ING)
$ git log --oneline
f596610 (HEAD -> main) Commit 1: New feature implementation
81fb37d Initial commit for exp-8
4a92531 (tag: v1.0, origin/main, origin/HEAD, master) Add Git manual experimen
t 5 execution to README
3c64823 Add experiment note to remote_work.txt using GitHub
b1b66a2 Add file for Experiment 5 push
5ee548e New folder added
a4ba270 Create README.md
9f16891 Initial commit

user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main|CHERRY-PICKING)
$ cat file.txt
<<<<<<< HEAD
Initial content.
=======
BUG: Fixed critical error.
>>>>>>> 2c476a3 (Commit 2: Critical bug fix for file.txt)
```

Experiment 9: Analysing and Changing Git History - Viewing Commit Details

- **Goal:** Inspect exactly what happened in a past save.
- **Command:** `git show <commit_id>` — Shows the author, date, and exact file changes for a specific commit.
- **Note:** Every commit has a unique **ID** (a string of letters and numbers); this ID is the key to finding any past version of our work.



Experiment 10: - Analysing and Changing Git History - Filtering History by Author and Date

- **Goal:** Find work done by a specific person during a specific time.
- **Command:** `git log --author="JohnDoe" --since="2023-01-01" --until="2023-12-31"` (syntax based on general Git usage; syllabus requires filtering by author and date range).
- **Note:** This is used by project managers to track contributions and audit changes over time.

```
user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git log --author="meghanamurali10" --since="2026-01-05" --until=
"2026-01-06"
commit f59661067cd17d4c5fbe522f06a89bd454ed262b (HEAD -> main)
Author: meghanamurali10-crypto <meghanamurali10@gmail.com>
Date:   Mon Jan 5 23:59:27 2026 +0530

    Commit 1: New feature implementation

commit 81fb37daa451af83e62279a59e51d1c48cce6dbf
Author: meghanamurali10-crypto <meghanamurali10@gmail.com>
Date:   Mon Jan 5 23:58:08 2026 +0530

    Initial commit for exp-8

commit 4a92531582c8873dfc6008bc436f3ce0a578c377 (tag: v1.0, origin
/main, origin/HEAD, master)
Author: meghanamurali10-crypto <meghanamurali10@gmail.com>
Date:   Mon Jan 5 23:30:52 2026 +0530

    Add Git manual experiment 5 execution to README
```

Experiment 11: - Analysing and Changing Git History - Displaying Recent History

- **Goal:** Quickly see the most recent activity.
- **Command:** `git log -n 5` — Displays only the **last five** commits in the history.
- **Note:** Limiting the log output makes it easier to read when a project has thousands of commits.

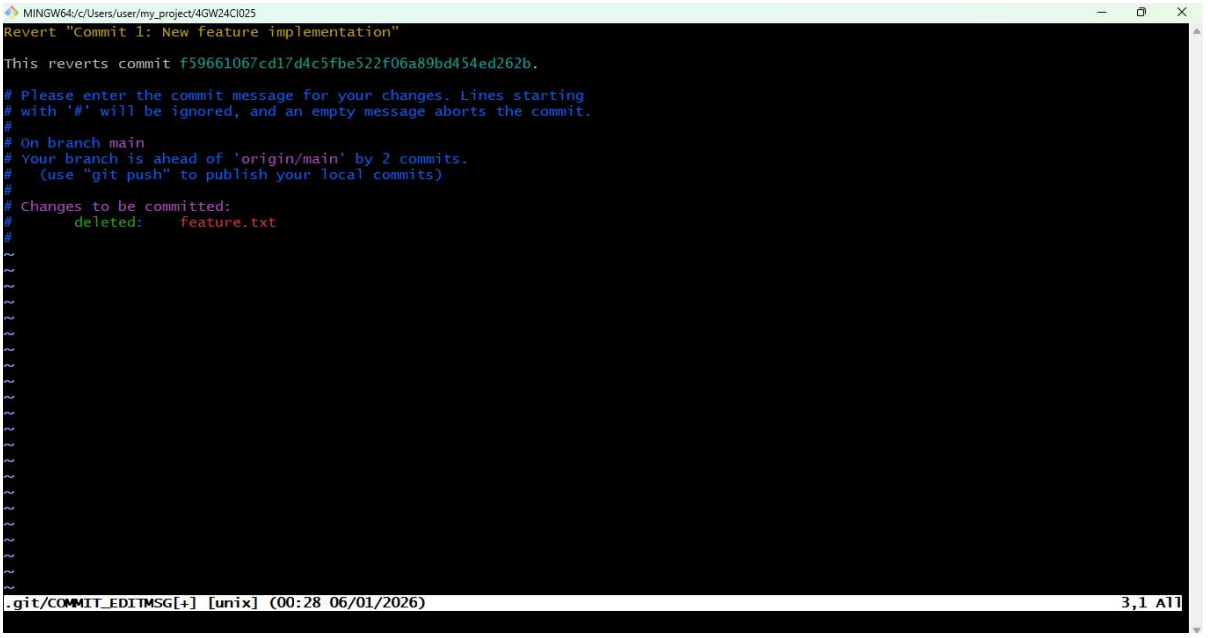
```
user@DESKTOP-M41VOET MINGW64 ~/my_project/4GW24CI025 (main)
$ git log -n 5
commit f59661067cd17d4c5fbe522f06a89bd454ed262b (HEAD -> main)
Author: meghanamurali10-crypto <meghanamurali10@gmail.com>
Date:   Mon Jan 5 23:59:27 2026 +0530

    Commit 1: New feature implementation

commit 81fb37daa451af83e62279a59e51d1c48cce6dbf
Author: meghanamurali10-crypto <meghanamurali10@gmail.com>
Date:   Mon Jan 5 23:58:08 2026 +0530

    Initial commit for exp-8

commit 4a92531582c8873dfc6008bc436f3ce0a578c377 (tag: v1.0, origin/main, origin/HEAD, master)
Author: meghanamurali10-crypto <meghanamurali10@gmail.com>
Date:   Mon Jan 5 23:30:52 2026 +0530

    Add Git manual experiment 5 execution to README

commit 3c648233fa9dcfff3a07a2ed09189cc588e8a060
Author: meghanamurali10-crypto <meghanamurali10@gmail.com>
Date:   Fri Dec 19 07:02:44 2025 +0530

    Add experiment note to remote_work.txt using GitHub

    Added a note about the experiment conducted on GitHub.

commit b1b66a2c77f2a34ed8fa0e4d7ab21e7c56eddda1
Author: meghanamurali10-crypto <meghanamurali10@gmail.com>
Date:   Thu Dec 18 22:07:37 2025 +0530

    Add file for Experiment 5 push
```

Experiment 12: - Analysing and Changing Git History - Undoing Changes

Goal: Fix a mistake safely.

Command: `git revert <commit_id>` — Undoes the changes of a specific commit by creating a **new** commit that does the opposite.

Note: `revert` is safer than deleting history because it leaves a record that a mistake was made and then fixed.




```
[main 236fb58] Revert "commit 1: New feature implementation"
 1 file changed, 1 deletion(-)
 delete mode 100644 feature.txt
```