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

TAMMISETTY HARINI
4GW24CI056

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

INDEX

SYLLABUS

1. **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.

2. **Creating and Managing Branches**

Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."

3. **Creating and Managing Branches**

Write the commands to stash your changes, switch branches, and then apply the stashed changes.

4. **Collaboration and Remote Repositories**

Clone a remote Git repository to your local machine.

5. **Collaboration and Remote Repositories**

Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

6. **Collaboration and Remote Repositories**

Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

7. **Git Tags and Releases**

Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.

8. Advanced Git Operations

Write the command to cherry-pick a range of commits from "source-branch" to the current.

9. Analyzing 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?

10. Analyzing and Changing Git History

Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."

11. Analyzing and Changing Git History

Write the command to display the last five commits in the repository's history.

12. Analyzing and Changing Git History

Write the command to undo the changes introduced by the commit with the ID "abc123".

Git Commands List

Git is a popular version control system used for tracking changes in software development projects. Here's a list of common Git commands along with brief explanations:

1. **git init**: Initializes a new Git repository in the current directory.
2. **git clone <repository URL>**: Creates a copy of a remote repository on your local machine.
3. **git add <file>**: Stages a file to be committed, marking it for tracking in the next commit.
4. **git commit -m "message"**: Records the changes you've staged with a descriptive commit message.
5. **git status**: Shows the status of your working directory and the files that have been modified or staged.
6. **git log**: Displays a log of all previous commits, including commit hashes, authors, dates, and commit messages.
7. **git diff**: Shows the differences between the working directory and the last committed version.
8. **git branch**: Lists all branches in the repository and highlights the currently checkedout branch.
9. **git branch <branchname>**: Creates a new branch with the specified name.
10. **git checkout <branchname>**: Switches to a different branch.
11. **git merge <branchname>**: Merges changes from the specified branch into the currently checked-out branch.

12. **git pull**: Fetches changes from a remote repository and merges them into the current branch.
13. **git push**: Pushes your local commits to a remote repository.
14. **git remote**: Lists the remote repositories that your local repository is connected to.
15. **git fetch**: Retrieves changes from a remote repository without merging them.
16. **git reset <file>**: Unstages a file that was previously staged for commit.
17. **git reset --hard <commit>**: Resets the branch to a specific commit, discarding all changes after that commit.
18. **git stash**: Temporarily saves your changes to a "stash" so you can switch branches without committing or losing your work.
19. **git tag**: Lists and manages tags (usually used for marking specific points in history, like releases).
20. **git blame <file>**: Shows who made each change to a file and when.
21. **git rm <file>**: Removes a file from both your working directory and the Git repository.
22. **git mv <oldfile> <newfile>**: Renames a file and stages the change.

These are some of the most common Git commands, but Git offers a wide range of features and options for more advanced usage. You can use git --help followed by the command name
to get more information about any specific command, e.g., git help commit.

EXPERIMENT – 1
GIT REPOSITORY INITIALIZATION AND FIRST COMMIT

Aim:

To initialize a Git repository and perform basic version control operations using Git commands.

Git is a distributed version control system used to track and manage changes in files. This experiment demonstrates the basic Git workflow such as repository creation, staging, and committing files.

Before starting version control operations, Git requires user identity configuration. In this step, the global user name and email are set, which will be linked to all commits created by the user.

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~ (master)
$ git init
Reinitialized existing Git repository in C:/Users/HP/.git/

HP@DESKTOP-KQ6FFS7 MINGW64 ~ (master)
$ git config --global user.name "Tammisetty Harini"

HP@DESKTOP-KQ6FFS7 MINGW64 ~ (master)
$ git config --global user.email "tammisettyharini@gmail.com"

HP@DESKTOP-KQ6FFS7 MINGW64 ~ (master)
$
```

Procedure:

Step 1: Create a New Directory

```
mkdir  GIT_MANUAL
 cd  GIT_MANUAL
```

A new directory is created to act as the project workspace where Git operations are performed.

Step 2: Initialize Git Repository

```
git init
```

This command initializes an empty Git repository by creating a hidden **.git** folder.

Step 3: Create a File

```
notepad file1.txt
```

A new text file is created to add content and demonstrate file tracking using Git.
For ex:

Step 4: Check Repository Status

```
git status
```

This command displays the current state of the repository and shows that the file is untracked.

Step 5: Add File to Staging Area

```
git add file1.txt
```

The file is added to the staging area, indicating that it is ready to be committed.

Step 6: Commit the File

```
git commit -m "Added file1.txt for first Git experiment"
```

This command permanently saves the staged changes into the local repository with

a commit message.

Step 7: View Commit History

```
git log
```

Displays the commit history including commit ID, author, date, and commit message.

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~ (master)
$ mkdir GIT_MANUAL

HP@DESKTOP-KQ6FFS7 MINGW64 ~ (master)
$ cd GIT_MANUAL

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git init
Initialized empty Git repository in C:/Users/HP/GIT_MANUAL/.git/

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ notepad file1.txt

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

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

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git add file1.txt

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git commit -m "Added file1.txt for first Git experiment"
[master (root-commit) 67a9770] Added file1.txt for first Git experiment
 1 file changed, 2 insertions(+)
 create mode 100644 file1.txt
```

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git add .

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git log
commit 67a9770ce7e92d05b9af900786f68e67858cbc79 (HEAD -> master)
Author: Tammisetty Harini <tammisettyharini@gmail.com>
Date:   Tue Jan 6 10:40:59 2026 +0530

    Added file1.txt for first Git experiment

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$
```

Result:

The Git repository was successfully created, and a file was added, staged, and committed using basic Git commands**.**

EXPERIMENT – 2:
Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."

Aim:
To create, list, switch, and merge branches in Git and understand the importance of branching.

Git branching allows developers to work on different versions of a project simultaneously without affecting the main code. This experiment demonstrates how to create and manage branches in a Git repository.

Procedure with Code and Step Description:

Step 1: Open Existing Git Repository

```
cd GIT_MANUAL
```

The existing Git repository from the previous experiment is opened to perform branch operations.

Step 2: Create a New Branch

```
git branch feature1
```

A new branch named **feature1** is created to develop new features independently.

Step 3: List Available Branches

```
git branch
```

This command displays all available branches and highlights the current active branch.

Step 4: Switch to the New Branch

```
git checkout feature1
```

The working directory is switched from the main branch to the **feature1** branch.

Step 5: Modify a File in New Branch

```
notepad file1.txt
```

Added Content:

```
This line is added in feature1 branch.
```

The file is modified to show how changes can be made independently in a separate branch.

Step 6: Add and Commit Changes

```
git add file1.txt
git commit -m "Updated file1.txt in feature1 branch"
```

The changes made in the new branch are staged and committed.

Step 7: Switch Back to Main Branch

```
git checkout master
```

The repository is switched back to the main branch to merge changes.

Step 8: Merge the Branch

```
git merge feature1
```

The changes from feature1 branch are merged into the main branch.

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git branch feature1

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git branch
  feature1
* master

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git checkout feature1
Switched to branch 'feature1'

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (feature1)
$ notepad file1.txt

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (feature1)
$ git add file1.txt

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (feature1)
$ git commit -m "Updated file1.txt in feature1 branch"
[feature1 d1a132b] Updated file1.txt in feature1 branch
 1 file changed, 1 insertion(+)

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (feature1)
$ git checkout master
Switched to branch 'master'

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git merge feature1
Updating 67a9770..d1a132b
Fast-forward
 file1.txt | 1 +
 1 file changed, 1 insertion(+)
```

Result:

A new branch was successfully created, modified, and merged into the main branch using Git commands.

EXPERIMENT – 3:
Write the commands to stash your changes, switch branches, and then apply the stashed changes.

Aim:
To create a remote repository on GitHub and perform push and pull operations between local and remote repositories.

A remote repository allows code to be stored and shared on a central server like GitHub. This experiment demonstrates how to connect a local Git repository to a remote repository and synchronize changes using push and pull commands.

Procedure with Code and Step Description:

Step 1: Check Current Status

```
git status
```

Displays the list of modified files that are not yet committed.

Step 2: Stash the Changes

```
git stash
```

Temporarily saves the uncommitted changes and cleans the working directory

Step 3: Switch to Another Branch

```
git checkout feature-branch
```

Switches from the current branch to another branch safely without losing changes.

Step 4: Apply the Stashed Changes

```
git stash apply
```

Re-applies the previously stashed changes to the current branch.

Step 5: View Stash List

```
git stash list
```

Displays all saved stashes.

Step 6: Remove Stash After Applying

```
git stash drop
```

Deletes the applied stash from the stash list.

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file1.txt


HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git stash
Saved working directory and index state WIP on master: 3d3c065 Updated file for experiment 3

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git checkout feature_branch
error: pathspec 'feature_branch' did not match any file(s) known to git

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git checkout -b feature_branch
Switched to a new branch 'feature_branch'
```

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (feature_branch)
$ git stash apply
On branch feature_branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file1.txt


HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (feature_branch)
$ git stash list
stash@{0}: WIP on master: 3d3c065 Updated file for experiment 3

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (feature_branch)
$ git add .
```

Result:

The local Git repository was successfully connected to GitHub, and push and pull operations were performed.

EXPERIMENT – 4:
Clone a remote Git repository to your local machine.

Aim:

To clone an existing remote repository into the local system using Git.

Cloning is the process of creating a local copy of an existing remote repository. This experiment demonstrates how a complete project along with its version history can be downloaded using Git clone.

Procedure:

Step 1: Open Git Bash

Git Bash is opened to execute Git commands.

Step 2: Select Directory for Cloning

```
cd Desktop
```

The directory where the remote repository will be cloned is selected.

Step 3: Copy Remote Repository URL

```
https://github.com/username/sample_repo.git
```

The repository URL is copied from the remote repository hosting platform.

Step 4: Clone the Remote Repository

```
git clone https://github.com/username/sample_repo.git
```

This command creates a local copy of the remote repository along with all files and commit history.

Step 6: Check Repository Status

```
git status
```

Displays the status of the cloned repository and confirms successful cloning.

Step 7: View Commit History

```
git log
```

Displays the commit history copied from the remote repository.

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git clone https://github.com/tammisettyharini-del/GIT_MANUAL.git
Cloning into 'GIT_MANUAL'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 13 (delta 2), reused 8 (delta 1), pack-reused 0 (from 0)
Receiving objects: 100% (13/13), done.
Resolving deltas: 100% (2/2), done.

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        GIT_MANUAL/

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

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git log
commit 3d3c0655a97133302ca6850448cf9f0326e9ef60 (HEAD -> master, origin/master, feature_branch)
Author: Tammisetty Harini <tammisettyharini@gmail.com>
Date:   Tue Jan 6 14:52:41 2026 +0530

    Updated file for experiment 3

commit d1a132b9a1d0ca005b480eb0d33860d07054e7a9 (feature1)
Author: Tammisetty Harini <tammisettyharini@gmail.com>
Date:   Tue Jan 6 11:32:52 2026 +0530

    Updated file1.txt in feature1 branch

commit 67a9770ce7e92d05b9af900786f68e67858cbc79
Author: Tammisetty Harini <tammisettyharini@gmail.com>
Date:   Tue Jan 6 10:40:59 2026 +0530

    Added file1.txt for first Git experiment

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$
```

Result:

The remote repository was successfully cloned into the local system using Git clone command.

EXPERIMENT – 5:
Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

Aim:
To fetch the latest changes from a remote repository and rebase the local branch onto the updated remote branch.

Git fetch is used to download the latest changes from a remote repository without merging them into the local branch. Git rebase is used to apply local commits on top of the updated remote branch to maintain a clean and linear commit history.

Procedure:
Step 1: Open the Existing Git Repository

```
cd git_exp1
```

The existing local Git repository is opened to perform collaboration operations with the remote repository.

Step 2: Check the Current Branch

```
git branch
```

Displays the list of branches and shows the currently active branch

Step 3: Fetch Latest Changes from Remote Repository

```
git fetch origin
```

Downloads the latest updates from the remote repository without affecting the

local branch.

Step 4: Check Repository Status

```
git status
```

Shows whether the local branch is behind the remote branch after fetching updates.

Step 5: Rebase Local Branch onto Remote Branch

```
git rebase origin/main
```

Reapplies local commits on top of the updated remote branch to keep commit history linear.

Step 6: Continue Rebase After Conflict Resolution (Optional)

```
git rebase –continue
```

Continues the rebase process after resolving conflicts, if any occur.

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git branch
  feature1
  feature_branch
* master

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git fetch origin

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   GIT_MANUAL


HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git rebase origin/master
Current branch master is up to date.

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$
```

EXPERIMENT – 6:
Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

Aim:
To merge a feature branch into the master branch while providing a custom commit message for the merge.

Git merge is used to combine changes from one branch into another branch. This experiment demonstrates how to merge a feature branch into the master branch using a custom commit message.

Procedure:

Step 1: Open the Existing Git Repository

```
cd git_exp1
```

The existing local Git repository is opened to perform merge operations.

Step 2: Switch to Master Branch

```
git checkout master
```

The working branch is switched to **master**, where the feature branch will be merged.

Step 3: Merge Feature Branch with Custom Commit Message

```
git merge feature-branch -m "Merged feature-branch into master with custom message"
```

This command merges the **feature-branch** into **master** and records the merge using a custom commit message.

Step 4: Verify Merge Status

```
git status
```

Displays the repository status and confirms that the merge was completed successfully.

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (feature_branch)
$ git checkout master
A       GIT_MANUAL
M       file1.txt
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git merge feature_branch -m "Merged feature-branch into master with custom message"
Already up to date.

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   GIT_MANUAL
        modified:   file1.txt


HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$
```

Result:
The feature branch was successfully merged into the master branch using a custom merge commit message.

EXPERIMENT – 7:
Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.

Aim
To create a lightweight Git tag named v1.0 for a commit in the local repository.

Git tags are used to mark specific points in a repository's history, usually for releases. A lightweight tag is a simple reference to a commit without additional metadata.

Procedure

Step 1: Open the Existing Git Repository

```
cd git_exp1
```

The existing local Git repository is opened to create a tag.

Step 2: View Commit History

```
git log –oneline
```

Displays the list of commits so that a specific commit can be identified for tagging.

Step 3: Create a Lightweight Git Tag

```
git tag v1.0
```

Creates a lightweight Git tag named **v1.0** for the latest commit in the local repository.

Step 4: Verify the Created Tag

```
git tag
```

Displays the list of tags created in the repository.

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git log --oneline
3d3c065 (HEAD -> master, origin/master, feature_branch) Updated file for experiment 3
d1a132b (feature1) Updated file1.txt in feature1 branch
67a9770 Added file1.txt for first Git experiment

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git tag v1.0

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git tag
v1.0

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$
```

Result:
A lightweight Git tag named **v1.0** was successfully created for a commit in the local repository.

EXPERIMENT – 8:
Write the command to cherry-pick a range of commits from "source-branch" to the current branch.

Aim:
To cherry-pick a range of commits from a source branch into the current branch using Git.

Git cherry-pick is used to apply specific commits from one branch to another branch. This experiment demonstrates how a range of commits can be selectively applied without merging the entire branch.

Procedure:

Step 1: Open the Existing Git Repository

```
cd git_exp1
```

The existing local Git repository is opened to perform advanced Git operations.

Step 2: Switch to the Target Branch

```
git checkout master
```

The branch where the selected commits need to be applied is checked out.

Step 3: View Commit History of Source Branch

```
git log --oneline source-branch
```

Displays the commit history of the source branch to identify the range of commits.

Step 4: Cherry-Pick a Range of Commits

```
git cherry-pick <start_commit>^..<end_commit>
```

Applies a specific range of commits from source-branch to the current branch.

Step 5: Verify the Applied Commits

```
git log --oneline
```

Verifies that the selected commits have been successfully applied to the current branch.

```
LENOVO@LAPTOP-AOD75M3F MINGW64 /d/gitManual (master)
$ git log --oneline --all
f31bfca (feature-branch) Initial commit : Add Readme.md file
440757e (origin/target-branch, target-branch) Stashed in target-branch
6c16f8c (refs/stash) WIP on master: 6f1d10d Changed from master branch to feature-branch.
cf90f9b index on master: 6f1d10d Changed from master branch to feature-branch.
6f1d10d (HEAD -> master, tag: v2.0, tag: v1.0, origin/master, origin/HEAD) Changed from master branch t
o feature-branch.
11f8cec Text.md file added with Experiment - 1.
1b3ac99 Text.md added with the contents of Experiment 1.
2bf03ed (tag: v3.0) Initial commit : Add Readme.md file

LENOVO@LAPTOP-AOD75M3F MINGW64 /d/gitManual (master)
$ git cherry-pick 2bf03ed
[master 15e2efa] Initial commit : Add Readme.md file
 Date: Thu Dec 18 11:41:44 2025 +0530
 1 file changed, 2 insertions(+)
 create mode 100644 Readme.md
```

Result:

Cherry-picking a range of commits was successfully performed, and advanced Git commit management was understood.

EXPERIMENT – 9:
Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?

Aim:
To view the details of a specific commit using its commit ID in Git.

Git allows users to inspect detailed information about any commit using its unique commit ID. This experiment demonstrates how to view commit details such as author, date, and commit message.

Procedure

Step 1: Open the Existing Git Repository

```
cd git_exp1
```

The existing local Git repository is opened to analyse commit history.

Step 2: View Details of a Specific Commit

```
git show <commit_id>
```

Displays detailed information about the specified commit, including author name, date, commit message, and changes made.

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git log --oneline
3d3c065 (HEAD -> master, tag: v1.0, origin/master) Updated file for experiment 3
d1a132b (feature1) Updated file1.txt in feature1 branch
67a9770 Added file1.txt for first Git experiment

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git show 67a9770
commit 67a9770ce7e92d05b9af900786f68e67858cbc79
Author: Tammisetty Harini <tammisettyharini@gmail.com>
Date:   Tue Jan 6 10:40:59 2026 +0530

    Added file1.txt for first Git experiment

diff --git a/file1.txt b/file1.txt
new file mode 100644
index 0000000..60f907f
--- /dev/null
+++ b/file1.txt
@@ -0,0 +1,2 @@
+Git Lab Experiment 1
+Learning basic Git commands.

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$
```

Result:
The details of the specified commit were successfully displayed using Git.

EXPERIMENT – 10:
Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."

Aim
To list all commits made by a specific author within a given date range using Git.

Git provides powerful filtering options to analyse commit history based on author and date. This experiment demonstrates how to list commits made by a specific author within a defined time period.

Procedure:

Step 1: List Commits by Author Between Given Dates

```
git log --author="JohnDoe" -- since="2023-01-01" --until="2023-12-31"
```

Displays all commits made by the author *JohnDoe* within the specified date range.

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git log --author="Tammisetty Harini" --since="2025-01-01" --until="2026-12-31"
commit 3d3c0655a97133302ca6850448cf9f0326e9ef60 (HEAD -> master, tag: v1.0, origin/master)
Author: Tammisetty Harini <tammisettyharini@gmail.com>
Date:   Tue Jan 6 14:52:41 2026 +0530

    Updated file for experiment 3

commit d1a132b9a1d0ca005b480eb0d33860d07054e7a9 (feature1)
Author: Tammisetty Harini <tammisettyharini@gmail.com>
Date:   Tue Jan 6 11:32:52 2026 +0530

    Updated file1.txt in feature1 branch

commit 67a9770ce7e92d05b9af900786f68e67858cbc79
Author: Tammisetty Harini <tammisettyharini@gmail.com>
Date:   Tue Jan 6 10:40:59 2026 +0530

    Added file1.txt for first Git experiment

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$
```

EXPERIMENT – 11:
Write the command to display the last five commits in the repository's history.

Aim
To display the last five commits in the Git repository history.

Git maintains a complete history of all commits made in a repository. This experiment demonstrates how to view the most recent commits using Git log options.

Procedure:

Step 2: Display the Last Five Commits

```
git log -5
```

Displays the most recent five commits along with commit ID, author, date, and commit message.

```
HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$ git log -3
commit 3d3c0655a97133302ca6850448cf9f0326e9ef60 (HEAD -> master, tag: v1.0, origin/master)
Author: Tammisetty Harini <tammisettyharini@gmail.com>
Date:   Tue Jan 6 14:52:41 2026 +0530

    Updated file for experiment 3

commit d1a132b9a1d0ca005b480eb0d33860d07054e7a9 (feature1)
Author: Tammisetty Harini <tammisettyharini@gmail.com>
Date:   Tue Jan 6 11:32:52 2026 +0530

    Updated file1.txt in feature1 branch

commit 67a9770ce7e92d05b9af900786f68e67858cbc79
Author: Tammisetty Harini <tammisettyharini@gmail.com>
Date:   Tue Jan 6 10:40:59 2026 +0530

    Added file1.txt for first Git experiment

HP@DESKTOP-KQ6FFS7 MINGW64 ~/GIT_MANUAL (master)
$
```

Result:
The last five commits in the repository history were successfully displayed.

EXPERIMENT – 12

Write the command to undo the changes introduced by the commit with the ID "abc123".

Aim:

To undo the changes introduced by a specific commit using its commit ID.

Git provides commands to safely undo changes introduced by previous commits. This experiment demonstrates how to revert a specific commit without affecting other commits.

Procedure:

Step 1: Undo Changes Introduced by a Specific Commit

```
git revert abc123
```

Creates a new commit that reverses the changes made by the commit with ID abc123**.**

Step 3: Verify Commit History

```
git log --oneline
```

Displays the commit history to confirm that the revert operation was successful.

```
GSSS@DESKTOP-G36BAK2 MINGW64 /e/New folder (10)
$ git init
Initialized empty Git repository in E:/New folder (10)/.git/

GSSS@DESKTOP-G36BAK2 MINGW64 /e/New folder (10) (master)
$ touch ok.txt && git add . && git commit -m "Stable"
[master (root-commit) d172c09] Stable
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 ok.txt

GSSS@DESKTOP-G36BAK2 MINGW64 /e/New folder (10) (master)
$ touch error.txt && git add . && git commit -m "Mistaken Commit"
[master c24267a] Mistaken Commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 error.txt

GSSS@DESKTOP-G36BAK2 MINGW64 /e/New folder (10) (master)
$ git log --oneline
c24267a (HEAD -> master) Mistaken Commit
d172c09 Stable

GSSS@DESKTOP-G36BAK2 MINGW64 /e/New folder (10) (master)
$ git revert c24267a
```

```
Revert "Mistaken Commit"

This reverts commit c24267ada7763436b67231c774c0e33d568d9288.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       deleted:    error.txt
#
```

Result:

The changes introduced by the specified commit were successfully undone using Git revert.

Appendix

Repository Link-

https://github.com/tammisettyharini-del/4GW24CI056.git