VISVESVARAYATECHNOLOGICALUNIVERSITY

“JnanaSangama”,Machhe,Belagavi,Karnataka-590018

Lab Experiment Record

Project Management with Git [BCSL358C]

SubmittedinpartialfulfillmenttowardsAECof3rdsemesterof

Bachelorof Engineering

in

ComputerScienceandEngineering

(ArtificialIntelligence&MachineLearning)

Submittedby

R VEENA

4GW24CI031

DEPARTMENTOF CSE(ArtificialIntelligence&MachineLearning)

GSSSINSTITUTEOFENGINEERING&TECHNOLOGYFORWOMEN

(AffiliatedtoVTU,Belagavi,ApprovedbyAICTE,NewDelhi&Govt.ofKarnataka)

K.R.SROAD,METAGALLI,MYSURU-570016,KARNATAKA

(AccreditedbyNAAC)

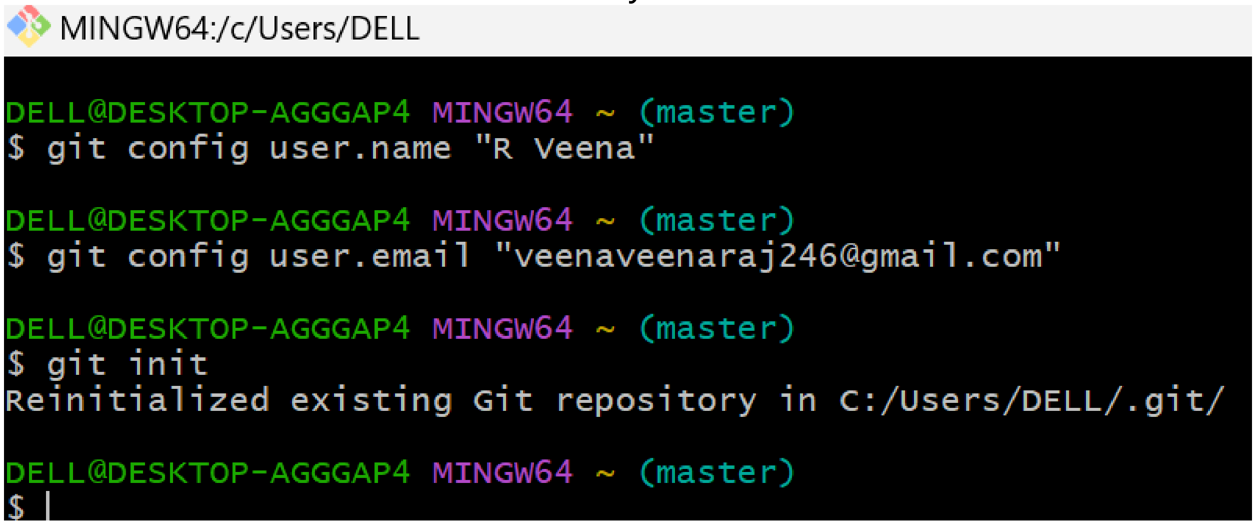
2025-2026

INDEX

EXPERIMENT–1: Git Repository Initialization and First Commit

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.



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~ (master)
$ git config user.name "R Veena"

DELL@DESKTOP-AGGGAP4 MINGW64 ~ (master)
$ git config user.email "veenaveenaraj246@gmail.com"

DELL@DESKTOP-AGGGAP4 MINGW64 ~ (master)
$ git init
Reinitialized existing Git repository in C:/Users/DELL/.git/

DELL@DESKTOP-AGGGAP4 MINGW64 ~ (master)
$ |
```

Procedure with code and step description :

Step 1 : Create a New Directory

```
mkdir git_exp1
cd git_exp1
```

Step 2 : Initialize Git Repository

```
git init
```

This command initializes an empty Git repository by creating a hidden –git folder to store version history.

Step 3 : Create a File

```
notepad file1.txt
```

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

```
Git Lab Experiment 1
Learning basic Git commands.
```

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
```

Step 6 : Commit the File

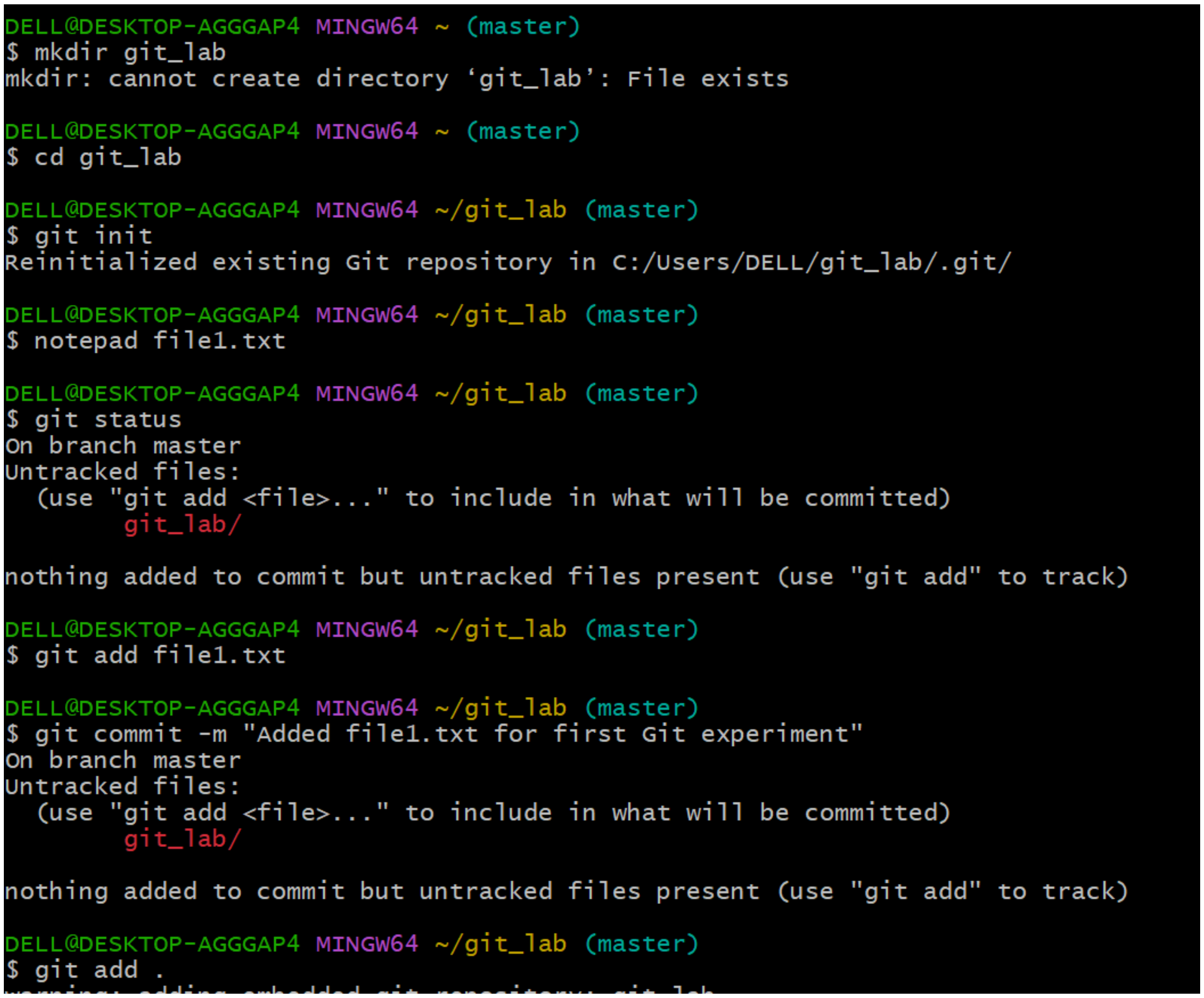
```
git commit -m "Added file1.txt for first Git experiment"
```

This command permanently saves the staged changes into the local repository with a commit message

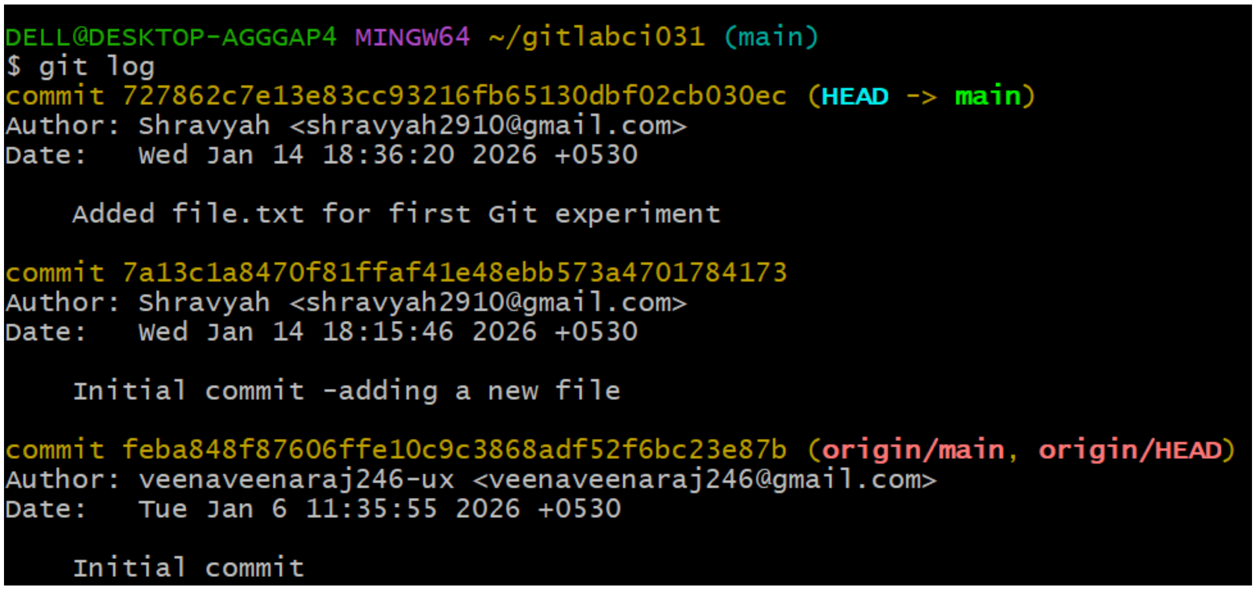
Step 7 : View Commit History

git log

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



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~ (master)
$ mkdir git_lab
mkdir: cannot create directory 'git_lab': File exists

DELL@DESKTOP-AGGGAP4 MINGW64 ~ (master)
$ cd git_lab

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git init
Reinitialized existing Git repository in C:/Users/DELL/git_lab/.git/

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ notepad file1.txt

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        git_lab/

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

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git add file1.txt

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git commit -m "Added file1.txt for first Git experiment"
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        git_lab/

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

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git add .
```



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~/gitlabci031 (main)
$ git log
commit 727862c7e13e83cc93216fb65130dbf02cb030ec (HEAD -> main)
Author: Shravyah <shravyah2910@gmail.com>
Date:   Wed Jan 14 18:36:20 2026 +0530

    Added file.txt for first Git experiment

commit 7a13c1a8470f81ffaf41e48ebb573a4701784173
Author: Shravyah <shravyah2910@gmail.com>
Date:   Wed Jan 14 18:15:46 2026 +0530

    Initial commit -adding a new file

commit feba848f87606ffe10c9c3868adf52f6bc23e87b (origin/main, origin/HEAD)
Author: veenaveenaraj246-ux <veenaveenaraj246@gmail.com>
Date:   Tue Jan 6 11:35:55 2026 +0530

    Initial commit
```

EXPERIMENT – 2: Creating And Managing Branches

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

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_exp1
```

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
```

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

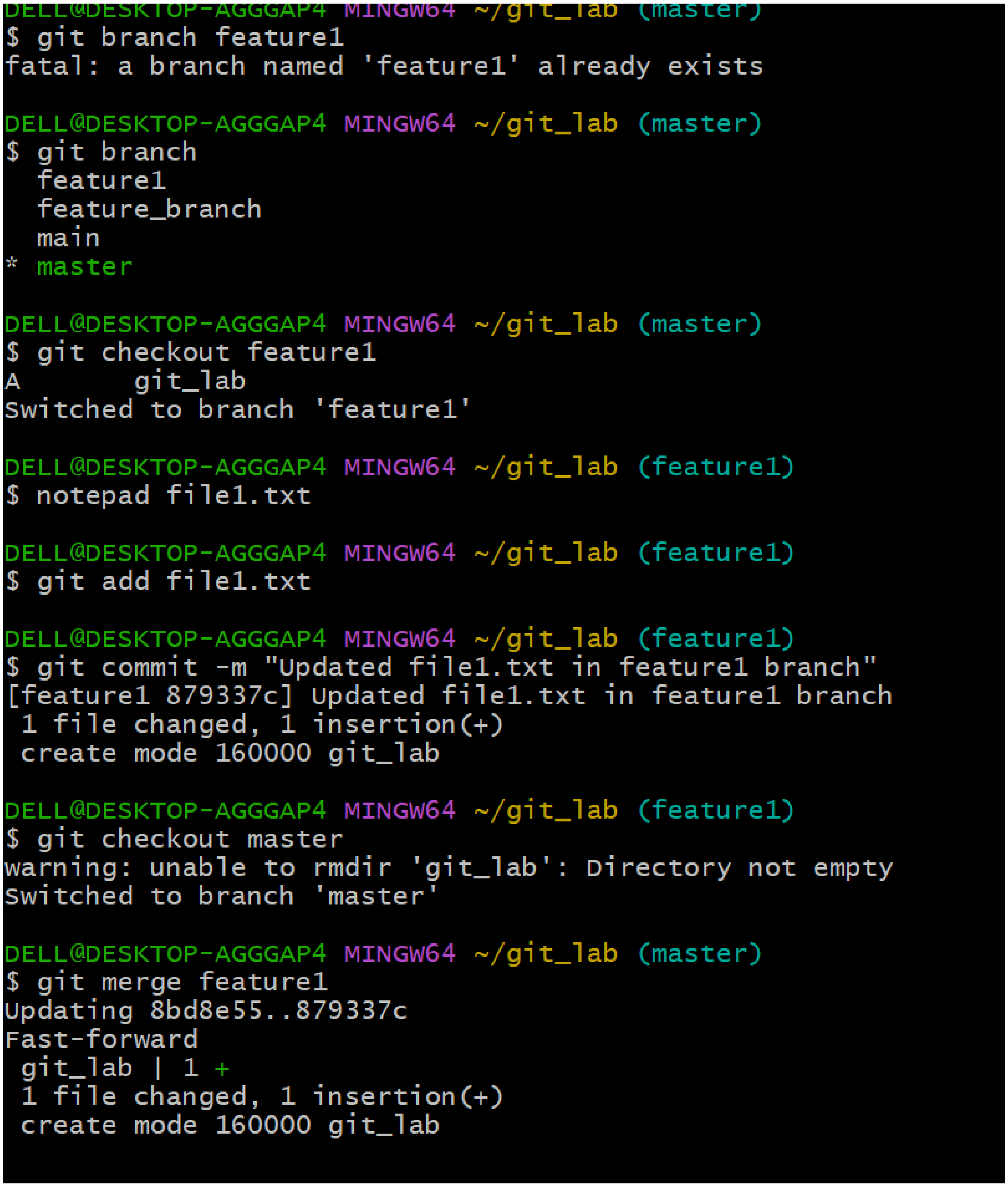
```
git checkout master
```

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.



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git branch feature1
fatal: a branch named 'feature1' already exists

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git branch
  feature1
  feature_branch
  main
* master

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git checkout feature1
A       git_lab
Switched to branch 'feature1'

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature1)
$ notepad file1.txt

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature1)
$ git add file1.txt

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature1)
$ git commit -m "Updated file1.txt in feature1 branch"
[feature1 879337c] Updated file1.txt in feature1 branch
 1 file changed, 1 insertion(+)
 create mode 160000 git_lab

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature1)
$ git checkout master
warning: unable to rmdir 'git_lab': Directory not empty
Switched to branch 'master'

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git merge feature1
Updating 8bd8e55..879337c
Fast-forward
 git_lab | 1 +
 1 file changed, 1 insertion(+)
 create mode 160000 git_lab
```

EXPERIMENT – 3: Creating and Managing Branches

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

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 without losing changes

Step 4 : Apply the stashed changes

```
git stash apply
```

Restores the previously stashed changes in the current branch.

Step 5 : View stash list

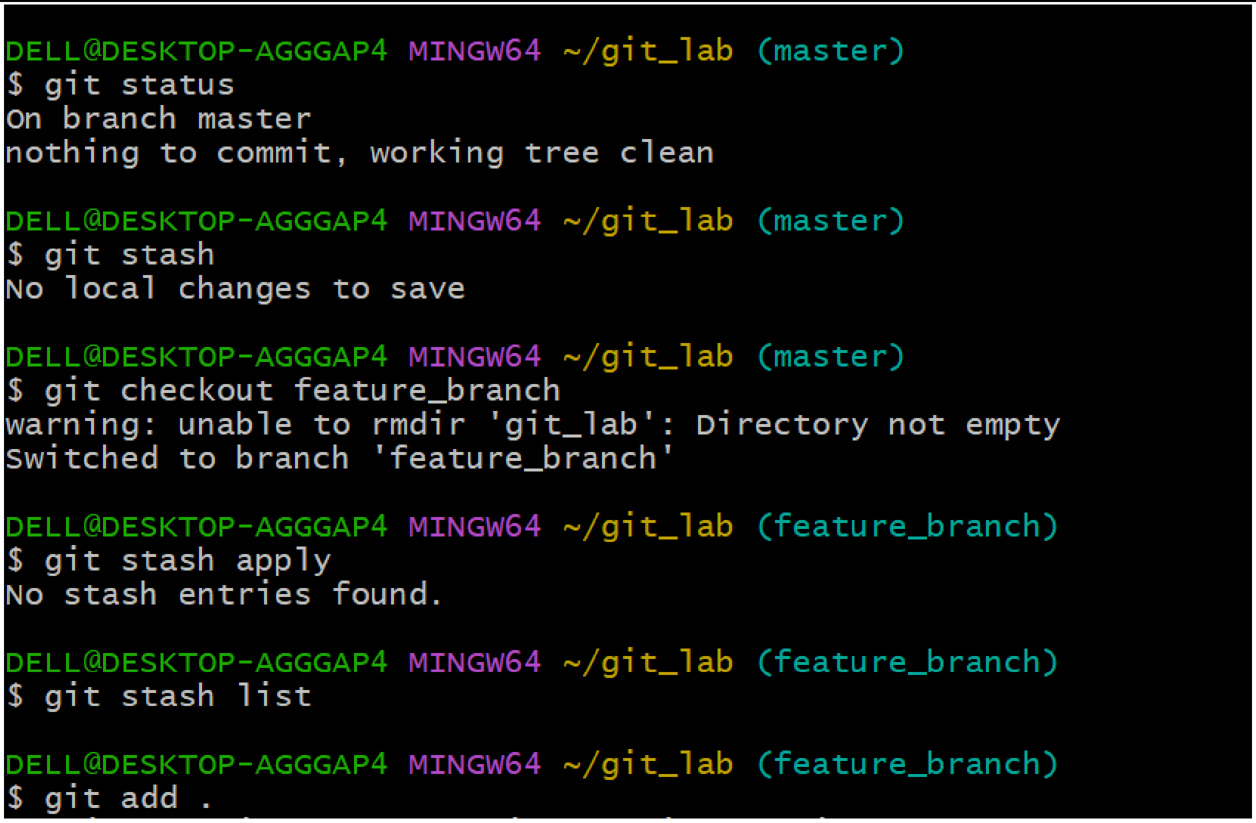
```
git stash list
```

Displays all saved stashes.

Step 6 : Remove stash after applying

```
git stash drop
```

Deletes the applied stash from the stash list.



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git status
On branch master
nothing to commit, working tree clean

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git stash
No local changes to save

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git checkout feature_branch
warning: unable to rmdir 'git_lab': Directory not empty
Switched to branch 'feature_branch'

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature_branch)
$ git stash apply
No stash entries found.

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature_branch)
$ git stash list

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature_branch)
$ git add .
```

EXPERIMENT – 4: Collaboration And Remote Repositories

Clone a remote Git repository to your local machine.

Cloning is the process of creating a local copy of an existing remote repository. This experiment demonstrates how Git clone is used to download a complete project along with its version history**.**

Procedure with code and step description :

Step 1 : Open Git Bash

Git Bash is opened to execute Git commands.

Step 2 : Select a 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/git_exp3.git

Step 4 : Clone the Remote Repository

```
git clone https://github.com/username/git_exp3.git
```

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

Step 5 : Check Repository Status

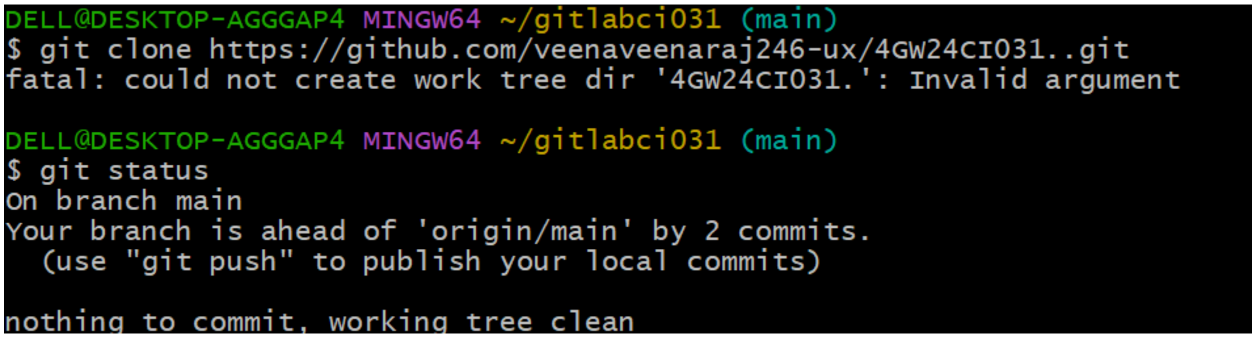
    git status

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

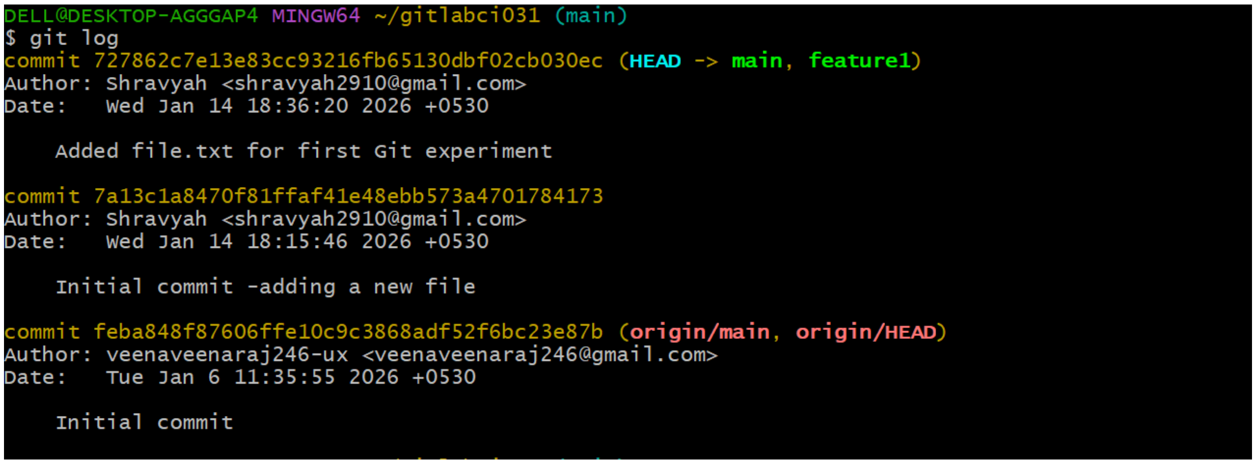
Step 6 : View Commit History

    git log

Shows the complete commit history that was cloned from the remote repository.



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~/gitlabci031 (main)
$ git clone https://github.com/veenaveenaraj246-ux/4GW24CI031..git
fatal: could not create work tree dir '4GW24CI031.': Invalid argument

DELL@DESKTOP-AGGGAP4 MINGW64 ~/gitlabci031 (main)
$ git status
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
```



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~/gitlabci031 (main)
$ git log
commit 727862c7e13e83cc93216fb65130dbf02cb030ec (HEAD -> main, feature1)
Author: Shravyah <shravyah2910@gmail.com>
Date:   Wed Jan 14 18:36:20 2026 +0530

    Added file.txt for first Git experiment

commit 7a13c1a8470f81ffaf41e48ebb573a4701784173
Author: Shravyah <shravyah2910@gmail.com>
Date:   Wed Jan 14 18:15:46 2026 +0530

    Initial commit -adding a new file

commit feba848f87606ffe10c9c3868adf52f6bc23e87b (origin/main, origin/HEAD)
Author: veenaveenaraj246-ux <veenaveenaraj246@gmail.com>
Date:   Tue Jan 6 11:35:55 2026 +0530

    Initial commit
```

EXPERIMENT – 5: Collaboration and Remote Repositories

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

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

Procedure :

Step 1 : Open the Existing Git Repository

```
cd git_exp1
```

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

Step 2 : Check Current Branch

```
git branch
```

Displays the list of working 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 merging them into 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

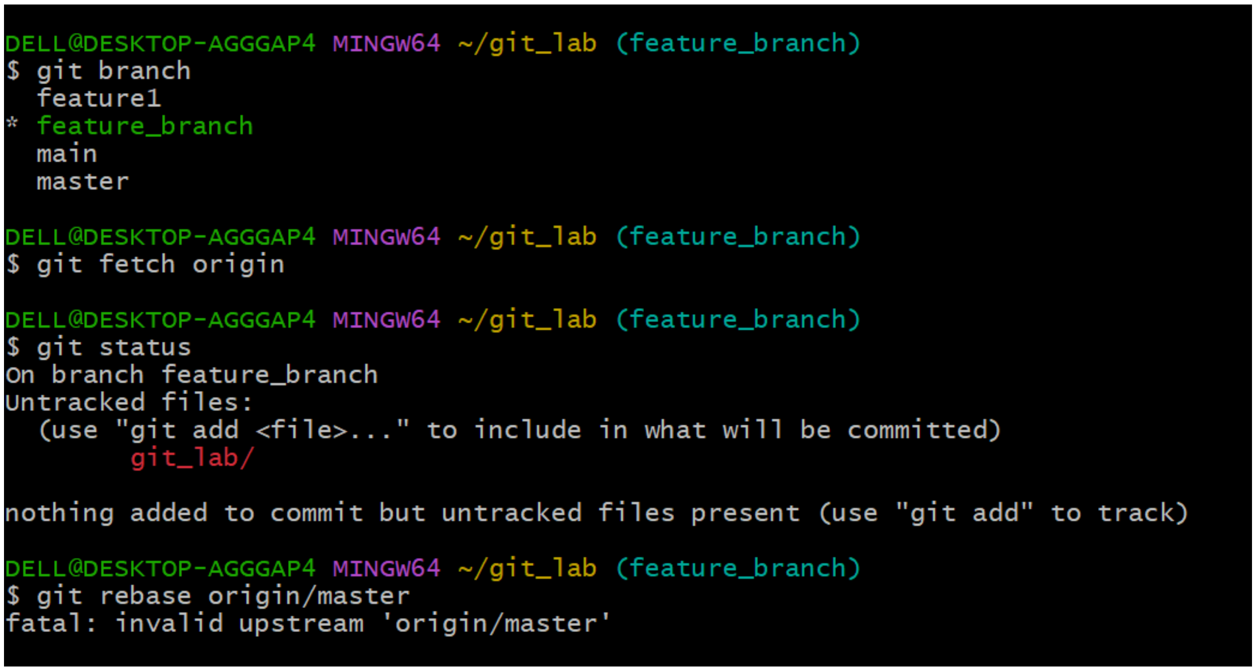
```
git rebase origin/main
```

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.



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature_branch)
$ git branch
  feature1
* feature_branch
  main
  master

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature_branch)
$ git fetch origin

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature_branch)
$ git status
On branch feature_branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        git_lab/

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

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature_branch)
$ git rebase origin/master
fatal: invalid upstream 'origin/master'
```

EXPERIMENT – 6 : Collaboration And Remote Repositories

Write the command to merge "feature-branch" into "master" 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

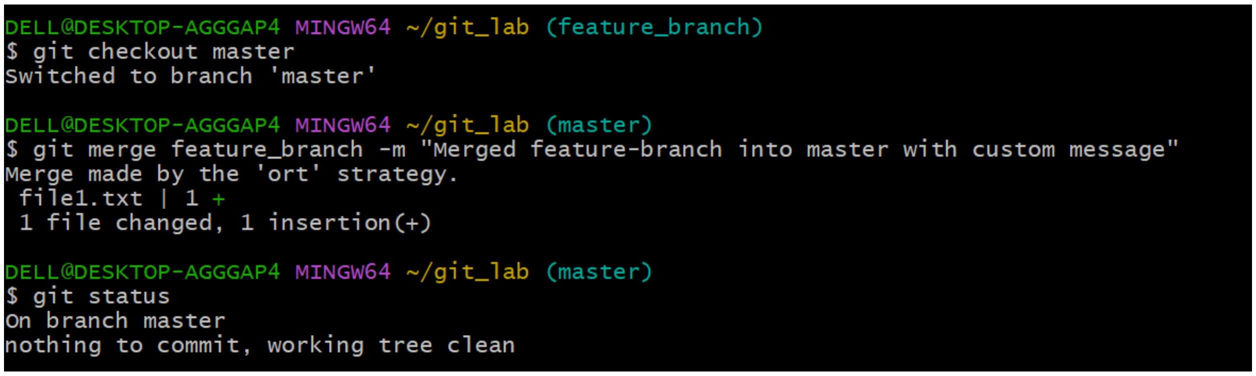
```
git merge feature-branch -m "Merged feature-branch into
master with custom message"
```

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.



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (feature_branch)
$ git checkout master
Switched to branch 'master'

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git merge feature_branch -m "Merged feature-branch into master with custom message"
Merge made by the 'ort' strategy.
 file1.txt | 1 +
 1 file changed, 1 insertion(+)

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git status
On branch master
nothing to commit, working tree clean
```

EXPERIMENT – 7 : Git Tags And Releases

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

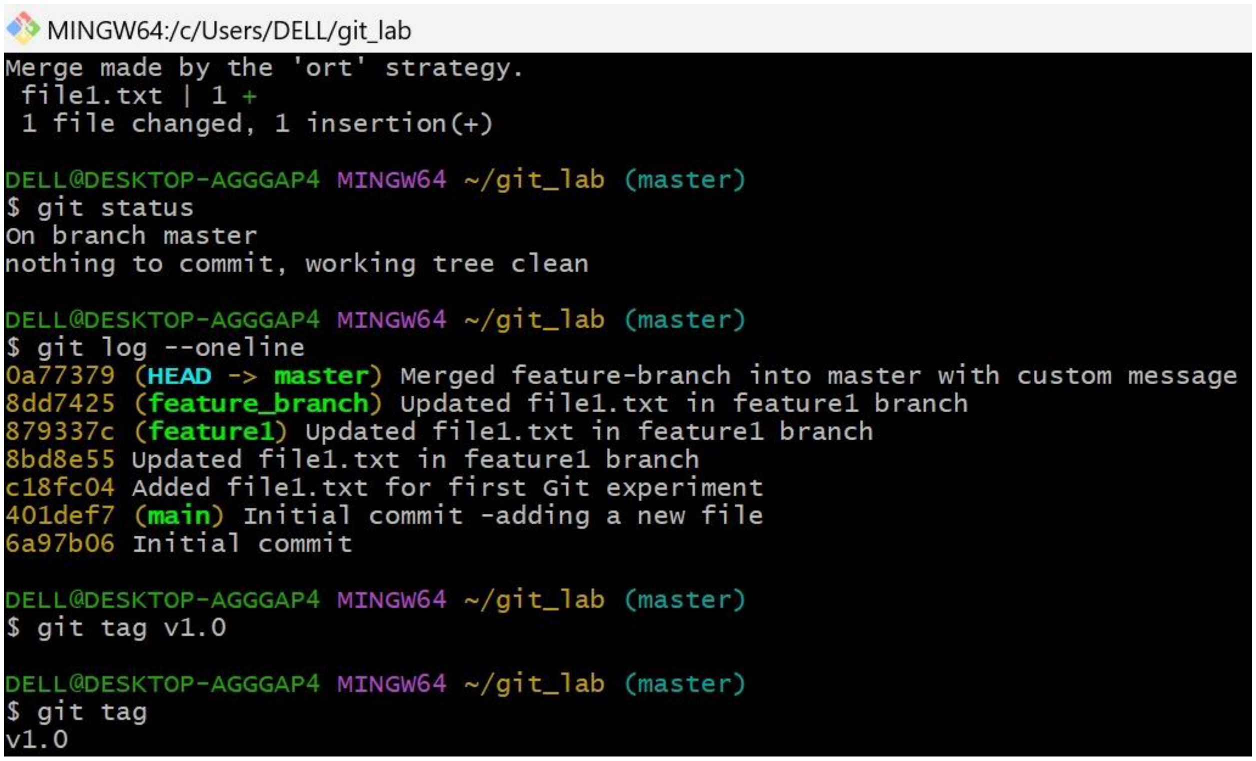
```
git tag v1.0
```

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.



MINGW64:/c/Users/DELL/git_lab

```
Merge made by the 'ort' strategy.
 file1.txt | 1 +
 1 file changed, 1 insertion(+)

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git status
On branch master
nothing to commit, working tree clean

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git log --oneline
0a77379 (HEAD -> master) Merged feature-branch into master with custom message
8dd7425 (feature_branch) Updated file1.txt in feature1 branch
879337c (feature1) Updated file1.txt in feature1 branch
8bd8e55 Updated file1.txt in feature1 branch
c18fc04 Added file1.txt for first Git experiment
401def7 (main) Initial commit -adding a new file
6a97b06 Initial commit

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git tag v1.0

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git tag
v1.0
```

EXPERIMENT – 8 : Advanced Git Operations

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

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

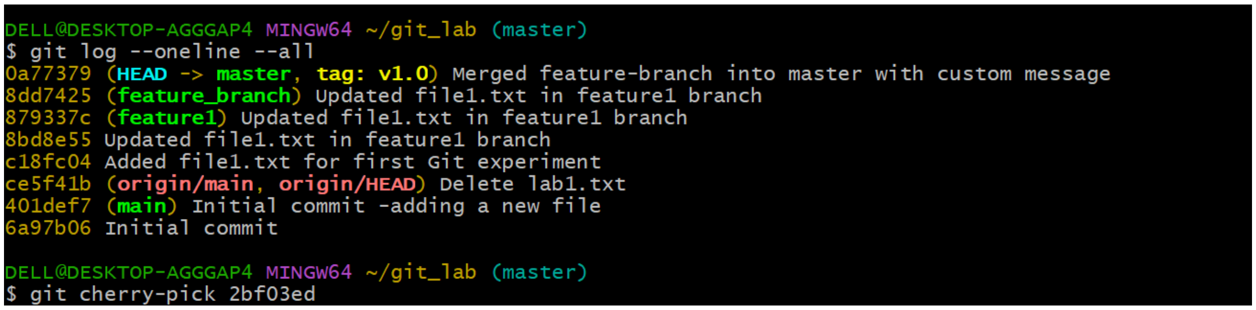
git cherry-pick <start_commit>^..<end_commit>

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.



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git log --oneline --all
0a77379 (HEAD -> master, tag: v1.0) Merged feature-branch into master with custom message
8dd7425 (feature_branch) Updated file1.txt in feature1 branch
879337c (feature1) Updated file1.txt in feature1 branch
8bd8e55 Updated file1.txt in feature1 branch
c18fc04 Added file1.txt for first Git experiment
ce5f41b (origin/main, origin/HEAD) Delete lab1.txt
401def7 (main) Initial commit -adding a new file
6a97b06 Initial commit

DELL@DESKTOP-AGGGAP4 MINGW64 ~/git_lab (master)
$ git cherry-pick 2bf03ed
```

EXPERIMENT – 9 : 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?

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

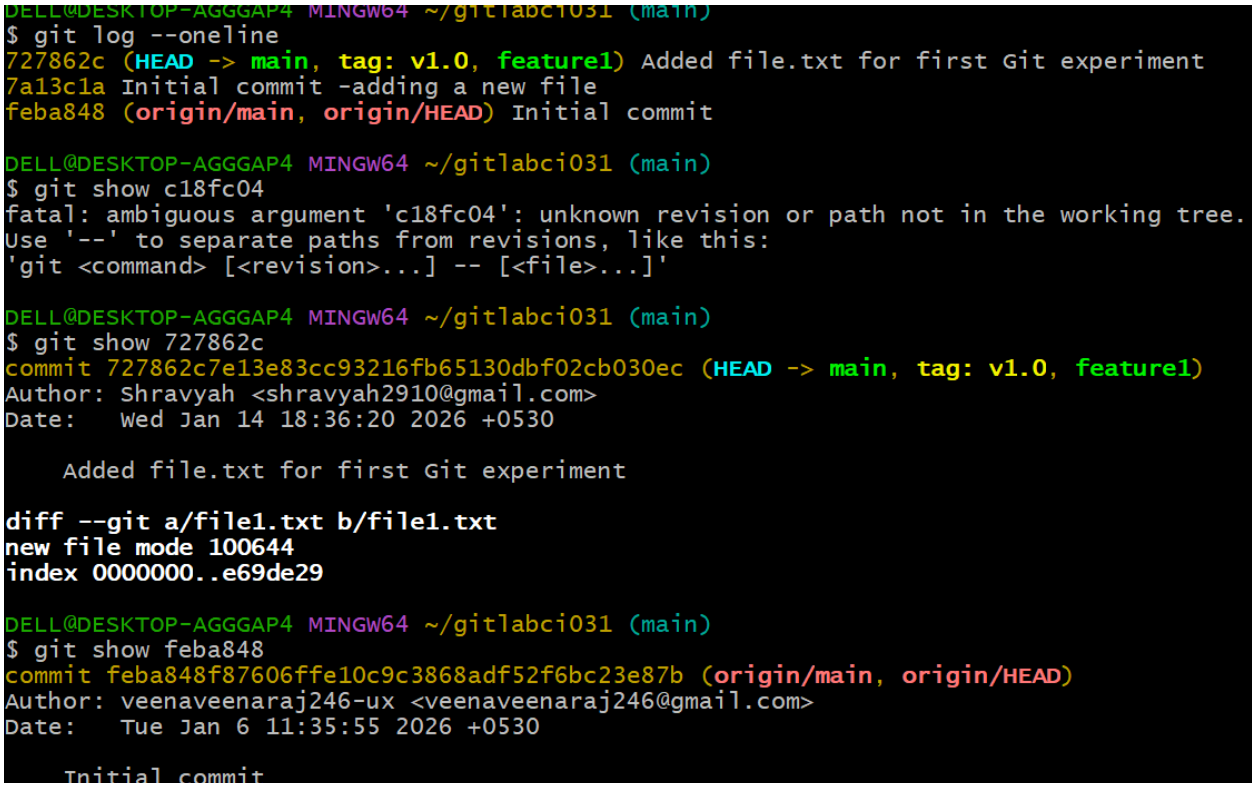
```
cd git_exp1
```

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.



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~/gitlabci031 (main)
$ git log --oneline
727862c (HEAD -> main, tag: v1.0, feature1) Added file.txt for first Git experiment
7a13c1a Initial commit -adding a new file
feba848 (origin/main, origin/HEAD) Initial commit

DELL@DESKTOP-AGGGAP4 MINGW64 ~/gitlabci031 (main)
$ git show c18fc04
fatal: ambiguous argument 'c18fc04': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'

DELL@DESKTOP-AGGGAP4 MINGW64 ~/gitlabci031 (main)
$ git show 727862c
commit 727862c7e13e83cc93216fb65130dbf02cb030ec (HEAD -> main, tag: v1.0, feature1)
Author: Shravyah <shravyah2910@gmail.com>
Date:   Wed Jan 14 18:36:20 2026 +0530

    Added file.txt for first Git experiment

diff --git a/file1.txt b/file1.txt
new file mode 100644
index 0000000..e69de29

DELL@DESKTOP-AGGGAP4 MINGW64 ~/gitlabci031 (main)
$ git show feba848
commit feba848f87606ffe10c9c3868adf52f6bc23e87b (origin/main, origin/HEAD)
Author: veenaveenaraj246-ux <veenaveenaraj246@gmail.com>
Date:   Tue Jan 6 11:35:55 2026 +0530

    Initial commit
```

EXPERIMENT – 10 : 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."

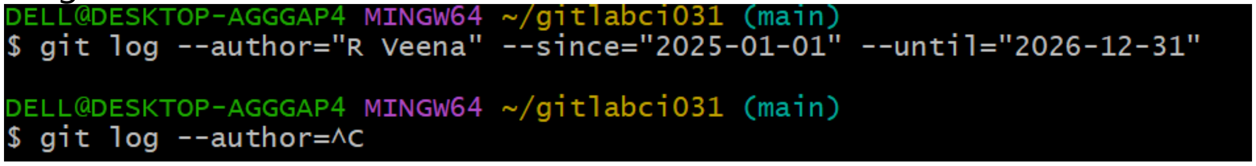
Git provides powerful filtering options to analyse commit history based on author and date. This experiment demonstrates how to list commits made by a specific author within a defined time period.

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.



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~/gitlabci031 (main)
$ git log --author="R Veena" --since="2025-01-01" --until="2026-12-31"

DELL@DESKTOP-AGGGAP4 MINGW64 ~/gitlabci031 (main)
$ git log --author=^C
```

EXPERIMENT – 11 : Analysing And Changing Git History

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

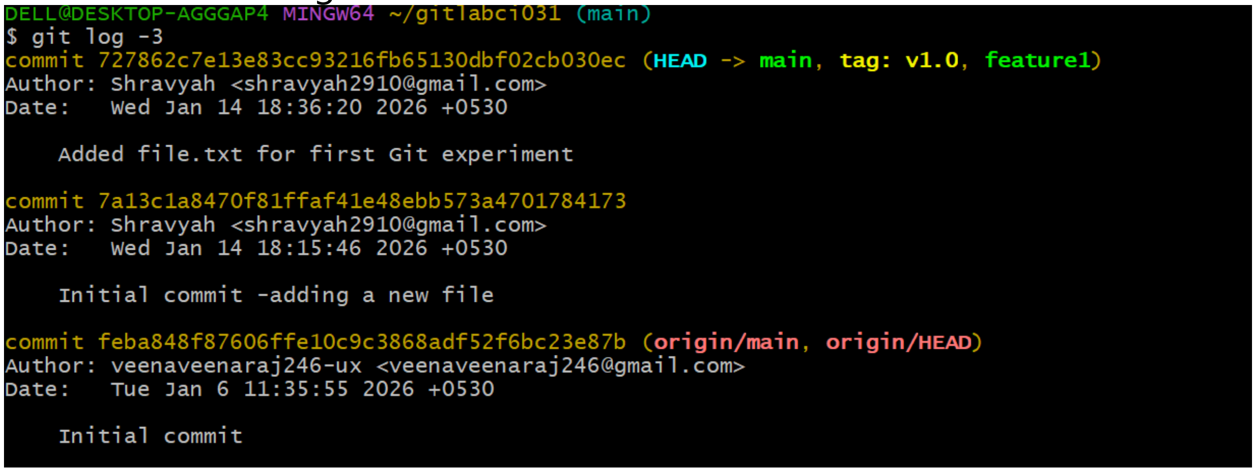
Git maintains a complete history of all commits made in a repository. This experiment demonstrates how to view the most recent commits using Git log options.

Procedure :

Step 1 : Display the Last Five Commits

```
git log -5
```

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



```
DELL@DESKTOP-AGGGAP4 MINGW64 ~/gitlabci031 (main)
$ git log -3
commit 727862c7e13e83cc93216fb65130dbf02cb030ec (HEAD -> main, tag: v1.0, feature1)
Author: Shravyah <shravyah2910@gmail.com>
Date:   Wed Jan 14 18:36:20 2026 +0530

    Added file.txt for first Git experiment

commit 7a13c1a8470f81ffaf41e48ebb573a4701784173
Author: Shravyah <shravyah2910@gmail.com>
Date:   Wed Jan 14 18:15:46 2026 +0530

    Initial commit -adding a new file

commit feba848f87606ffe10c9c3868adf52f6bc23e87b (origin/main, origin/HEAD)
Author: veenaveenaraj246-ux <veenaveenaraj246@gmail.com>
Date:   Tue Jan 6 11:35:55 2026 +0530

    Initial commit
```

EXPERIMENT – 12 : Analysing And Changing Git History

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

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

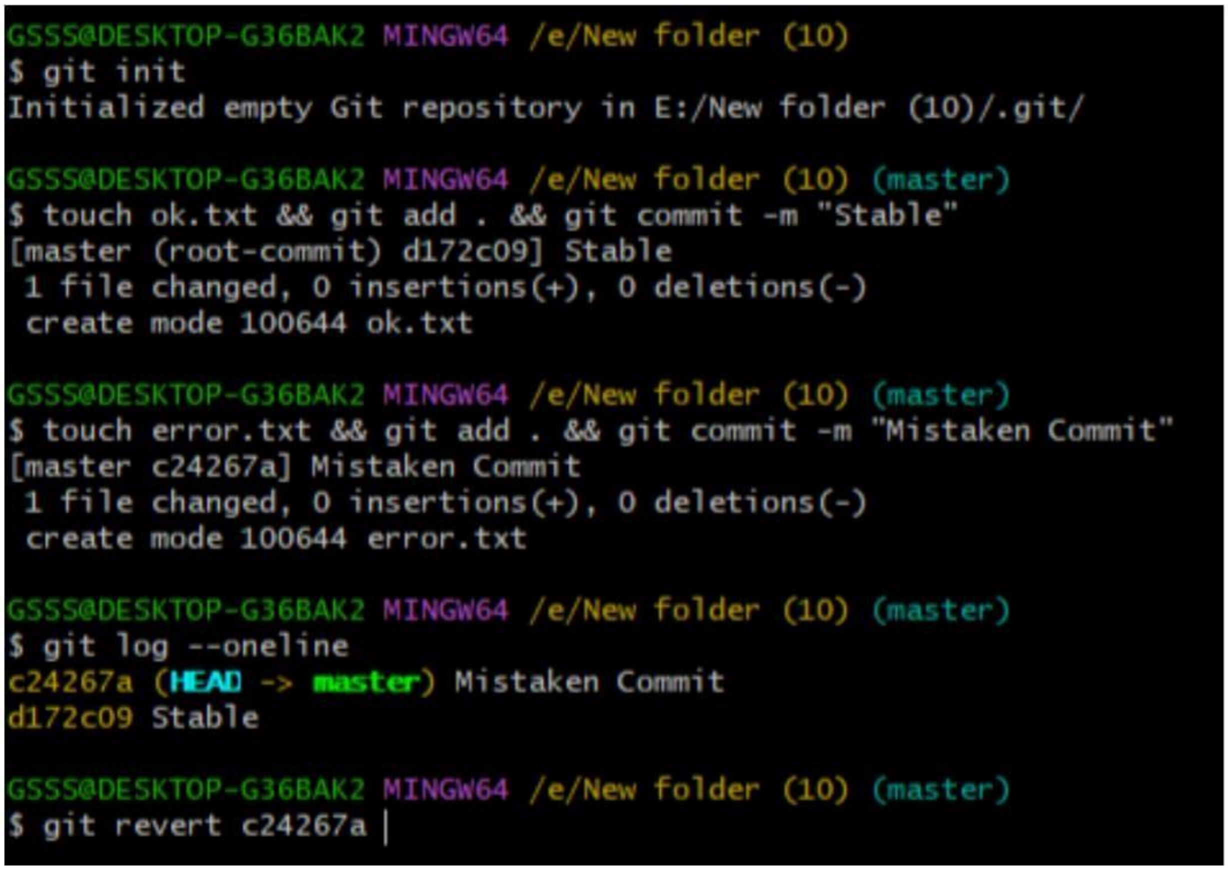
```
git revert abc123
```

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

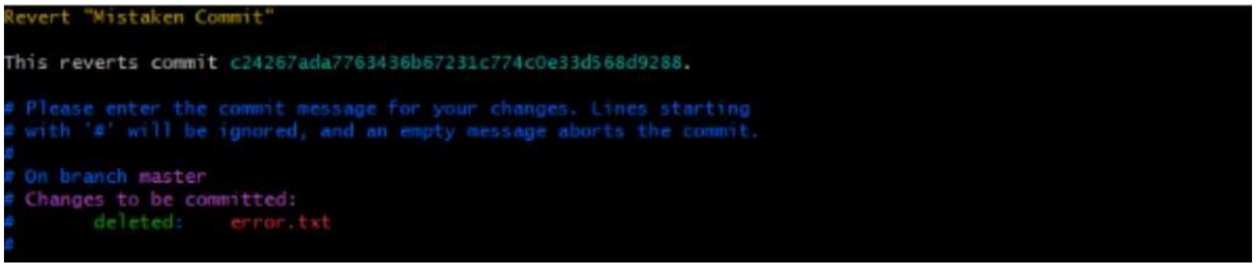
Step 2 : Verify Commit History

```
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
#
```

APPENDIX

Repository link

https://github.com/veenaveenaraj246-ux/4GW24CI031..git