**EXPERIMENT No. 01:** Setting Up and Basic Commands



|  |  |
| --- | --- |
| 1.1 Objective | 1.4 Introduction |
| 1.2 System Configuration | 1.5 Procedure and Results |
| 1.3 Pre-Requisite |  |
|  |  |

* 1. **Objective**

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

* 1. **System Configuration**

Windows 10, Linux,Mac

* 1. **Pre-Requisite**

Install Git bash and make required settings.

* 1. **Introduction**

git init

This command is used to initialize a new Git repository in the current directory. It sets up the necessary data structures and files that Git needs to start tracking changes to your project.

git clone ‘remote repository link’

This command is used to create a copy of an existing Git repository from a remote location (in this case, from the specified URL) to your local machine. It not only copies the files but also sets up a connection to the original repository so you can pull in updates later.

git add .

This command is used to stage all changes in the current directory for the next commit. Staging is the process of preparing files to be included in the next commit. The dot (.) indicates that all changes, including new files, modified files, and deleted files, should be staged.

git commit -m "<message>"

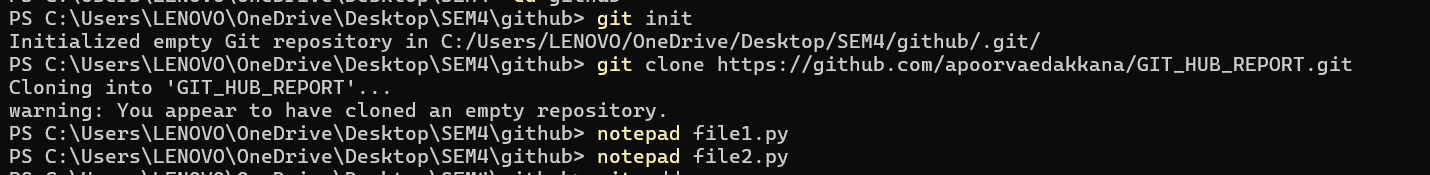
This command is used to record the changes that have been staged (using git add) in the repository. The -m flag allows you to include a short message that describes the changes made in this commit. It's good practice to write clear and concise commit messages that explain the purpose of the changes.

git push

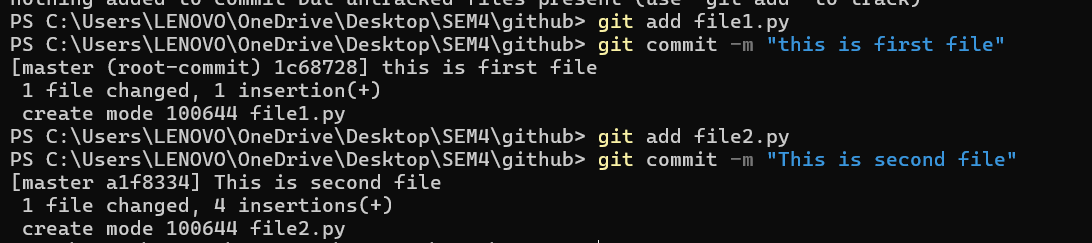
This command is used to upload local repository content to a remote repository. In the context of GitHub, it typically sends committed changes from your local repository to the remote repository on GitHub. This allows others to see the changes you've made and collaborate with you. Depending on your Git configuration, you may need to specify the remote repository and branch name, but if you've cloned a repository, Git usually sets up the default remote and branch for you.

* 1. **Procedure and Results**

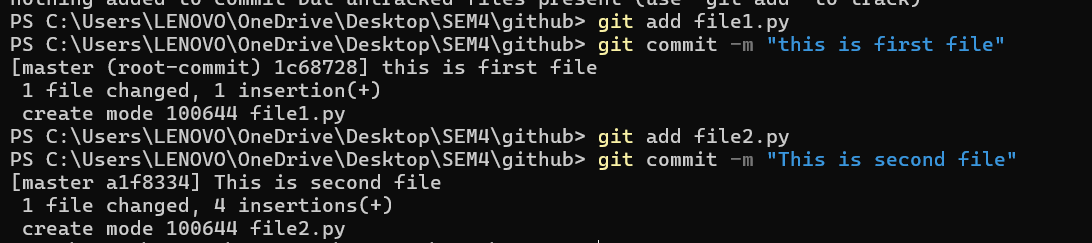
git clone ‘remote repository link’



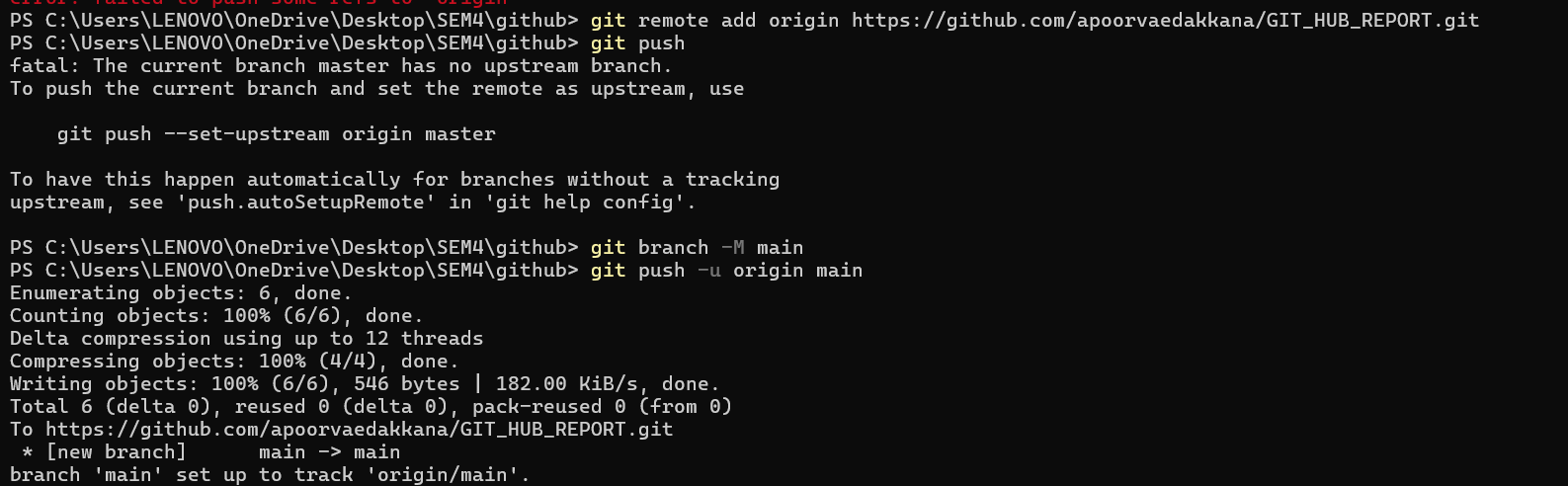
git add .



git commit -m "<message>"



git push



# EXPERIMENT No. 02: Creating and Managing Branches



|  |  |
| --- | --- |
| 2.1 Objective | 2.4 Introduction |
| 2.2 System Configuration | 2.5 Procedure and Results |
| 2.3 Pre-Requisite |  |
|  |  |

**2.1Objective**

1. Create a new branch named “feature-branch”. Switch to the “master” branch. Merge the “feature-branch” into “master”.
2. Write the commands to stash your changes, switch branches, and then apply the stashed changes.

**2.2 System Configuration**

Windows 10, Linux, Mac

**2.3 Pre-Requisite**

1. Git is installed on your system.

2. You have a Git repository initialized and have some changes made to the files.

3. You have at least two branches: master/main and feature-branch.

**2.4 Introduction**

. git branch -b feature-branch

This command creates a new branch named "feature-branch" but doesn't switch to it.

git checkout master/main

This command switches to the "master" branch.

git merge feature-branch

This command merges changes from "feature-branch" into the "master" branch git stash:

Stash your changes

git checkout feature-branch

Switch branches (example: from "master" to "feature-branch")

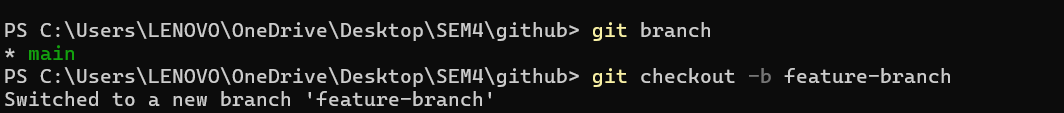
git stash apply

Apply the stashed changes

**2.5 Procedure and Results**

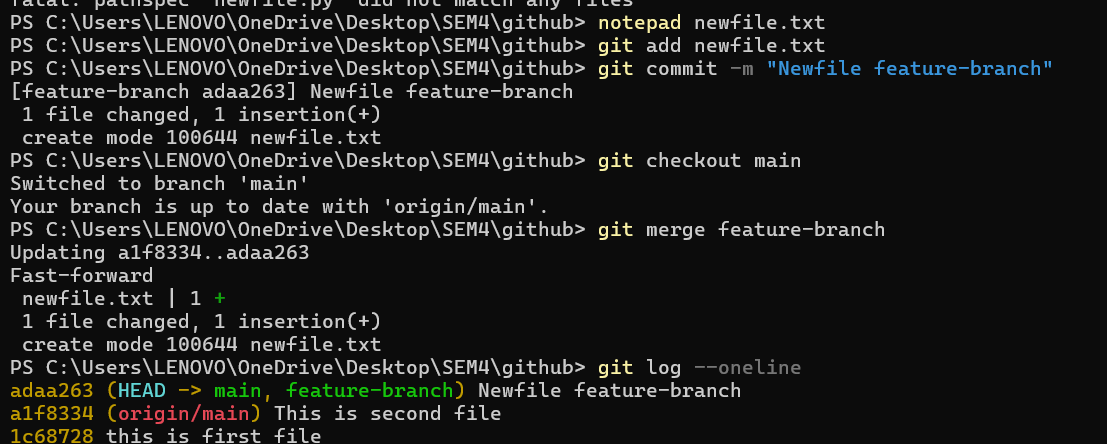
git branch feature-branch

git checkout feature-branch



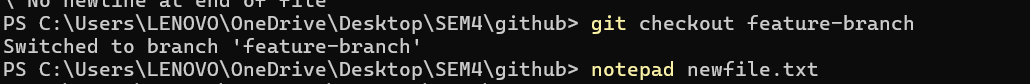
git add .

git commit -m “<message>” git checkout master git merge

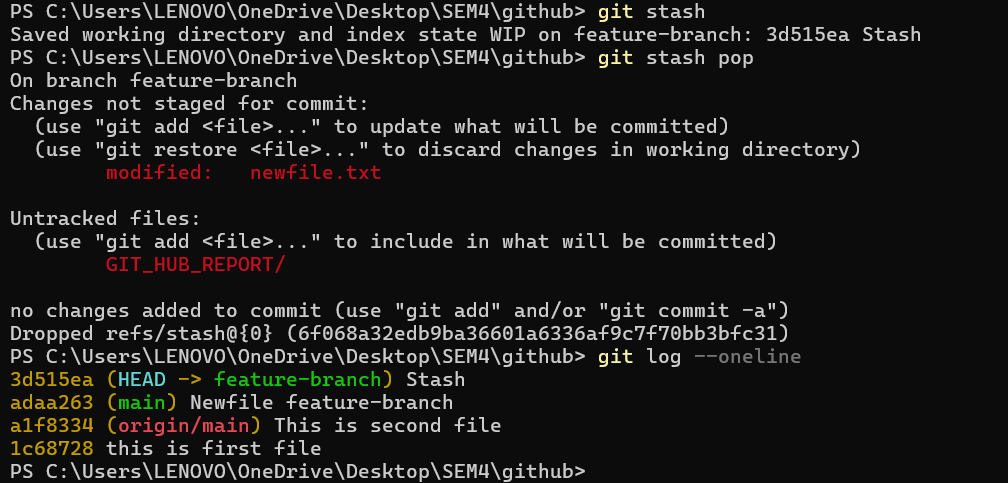


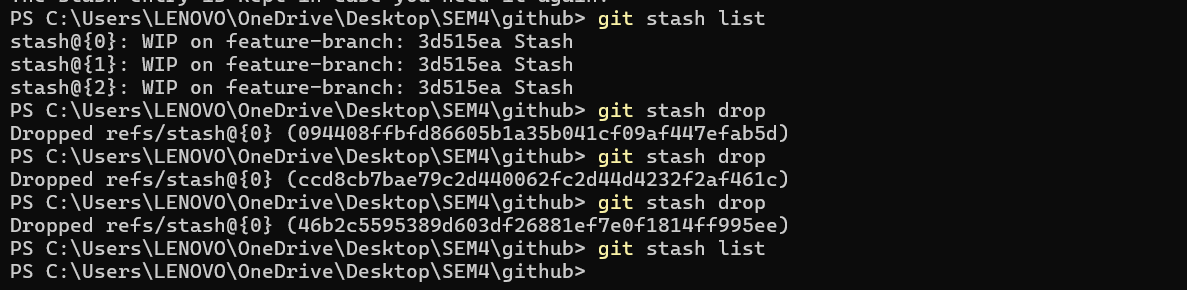
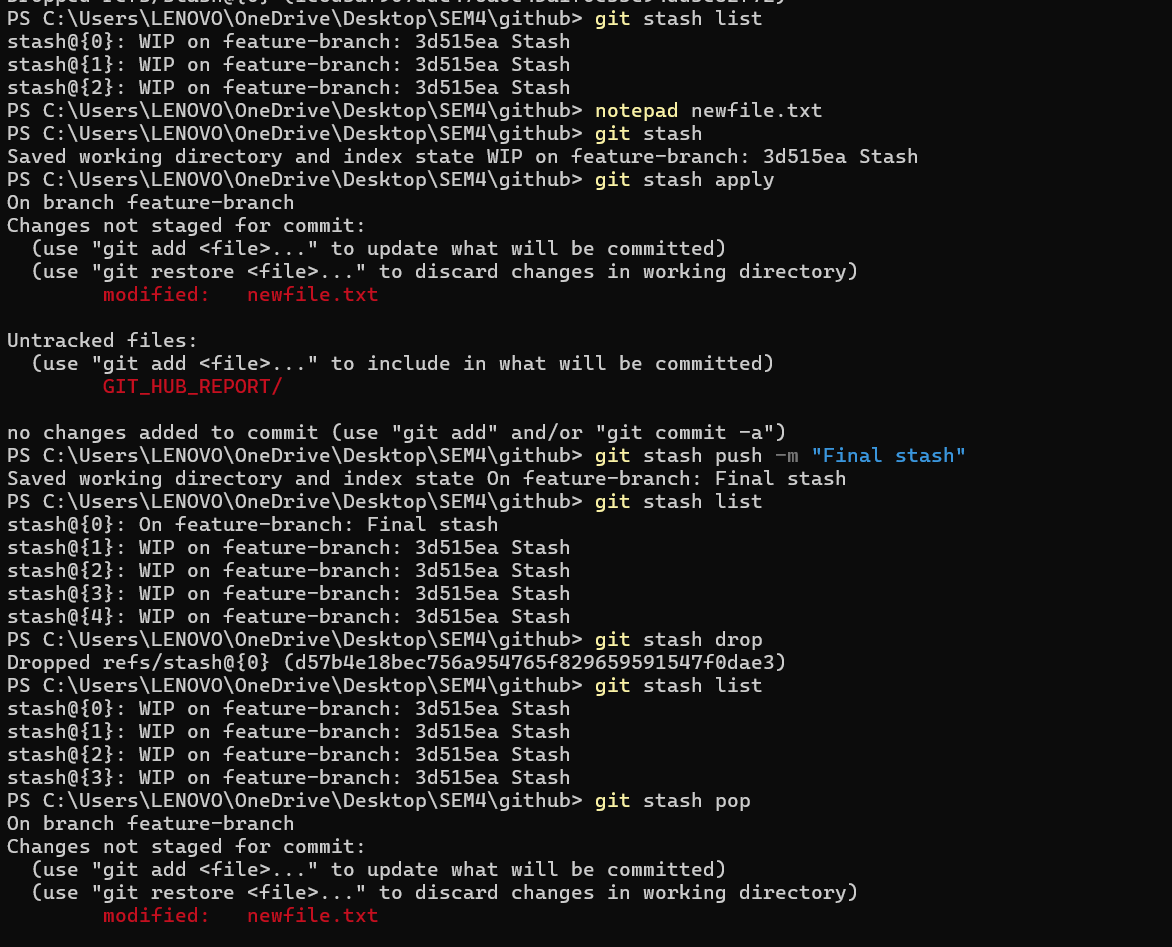
git stash

git checkout feature-branch



git stash apply





**EXPERIMENT No.03:** Collaboration and Remote Repositories



|  |  |
| --- | --- |
| 3.1 Objective | 3.4 Introduction |
| 3.2 System Configuration | 3.5 Procedure and Results |
| 3.3 Pre-Requisite |  |
|  |  |

**3.1 Objectives:**

* + 1. Clone a remote Git repository to your local machine.
    2. Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.
    3. Write a command to merge “feature-branch” into “master” while providing a custom commit message for the merge.
  1. **System Configuration:**

Windows 10

* 1. **Pre-Requisite:**

Git bash should be installed

Visual studio code editor should be installed

**3.4 Introduction:**

* + 1. Clone a remote Git repository to your local machine: git clone <repository\_url>

This command clones a remote Git repository onto your local machine, creating

a new directory with the same name as the repository.

* + 1. Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch:

git checkout -b feature-branch git fetch origin

git rebase origin/<branch\_name>

The first command (git fetch origin) fetches the latest changes from the remote repository named "origin" without merging them into your local branches. The second command (git rebase origin/<branch\_name>) rebases your current local branch onto the updated remote branch, integrating the latest changes from the remote repository while maintaining the commit history.

* + 1. Write a command to merge “feature-branch” into “master” while providing a custom commit message for the merge:

git checkout master

git merge --no-ff feature-branch -m "Custom commit message"

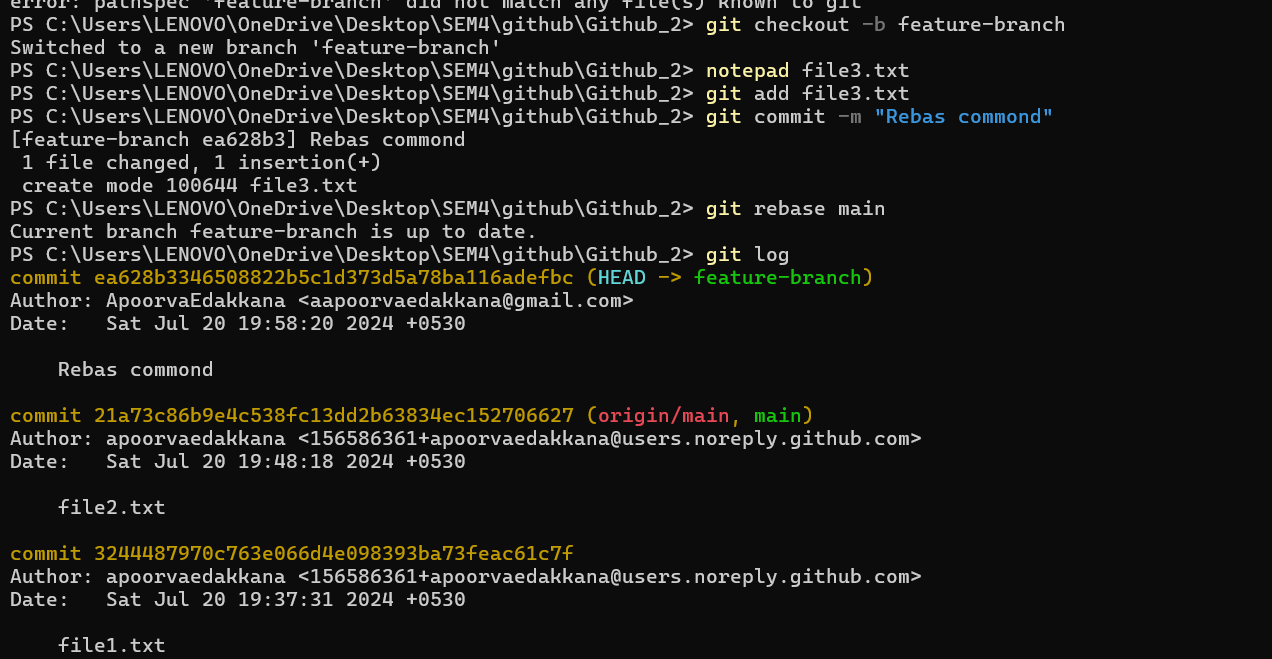
The first command (git checkout master) switches to the "master" branch. The second command (git merge --no-ff feature-branch -m "Custom commit message") merges the changes from the "feature-branch" into the "master" branch, creating a merge commit with the provided custom commit message. The --no-ff option ensures that a merge commit is always created, even if the merge could be performed with a fast-forward.

**3.5 Procedure and Results:**

git clone <repository\_url>

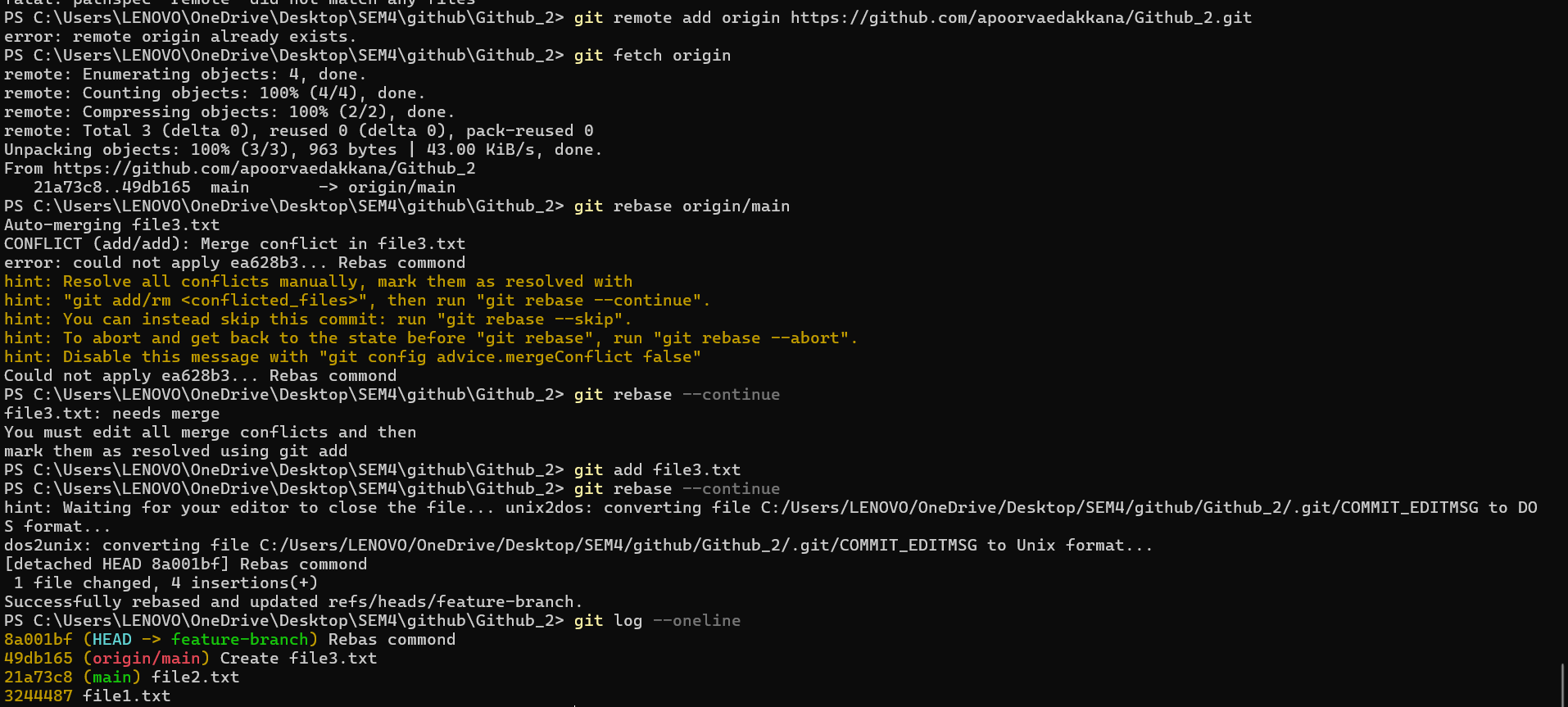


git checkout -b feature-branch



git fetch origin

git rebase origin/<branch\_name>



**EXPERIMENT No.04:** Git Tags and Releases

|  |  |
| --- | --- |
| 4.1 Objective | 4.4 Introduction |
| 4.2 System Configuration | 4.5 Procedure and Results |
| 4.3 Pre-Requisite |  |
|  |  |

**4.1 Objective**

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

**4.2 System Configuration**

Windows 10, Linux, Mac

**4.3 Pre-Requisite**

Install Git bash and make required settings.

**4.4 Introduction**

git log

to view the commit history and find the commit ID you want to tag. Each commit is identified by a unique hash. git

tag v1.0 <commit\_id>

git tag followed by the tag name (e.g., "v1.0") and the commit ID to create a lightweight tag. Lightweight tags are simply pointers to specific commits and do not contain additional metadata like annotated tags.

git show v1.0

Use git show followed by the tag name to verify that the tag was created correctly and is pointing to the desired commit. This command will display the details of the tag, including the commit it points to.

git push origin v1.0

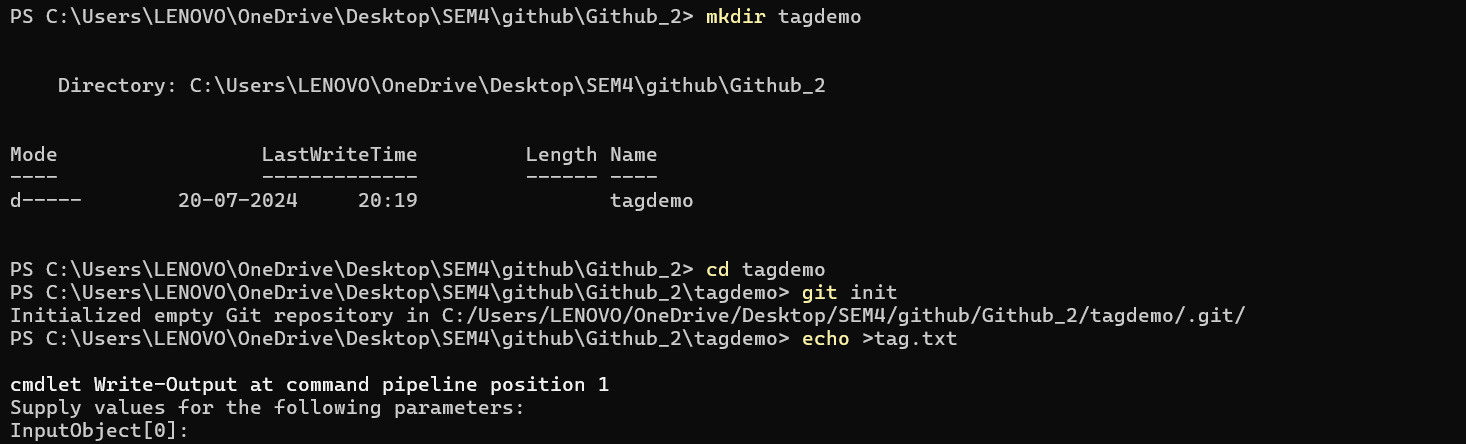
If you want to share the tag with others, you can push it to a remote repository using git push. This step is optional and depends on your workflow and the need to share the tag with others.

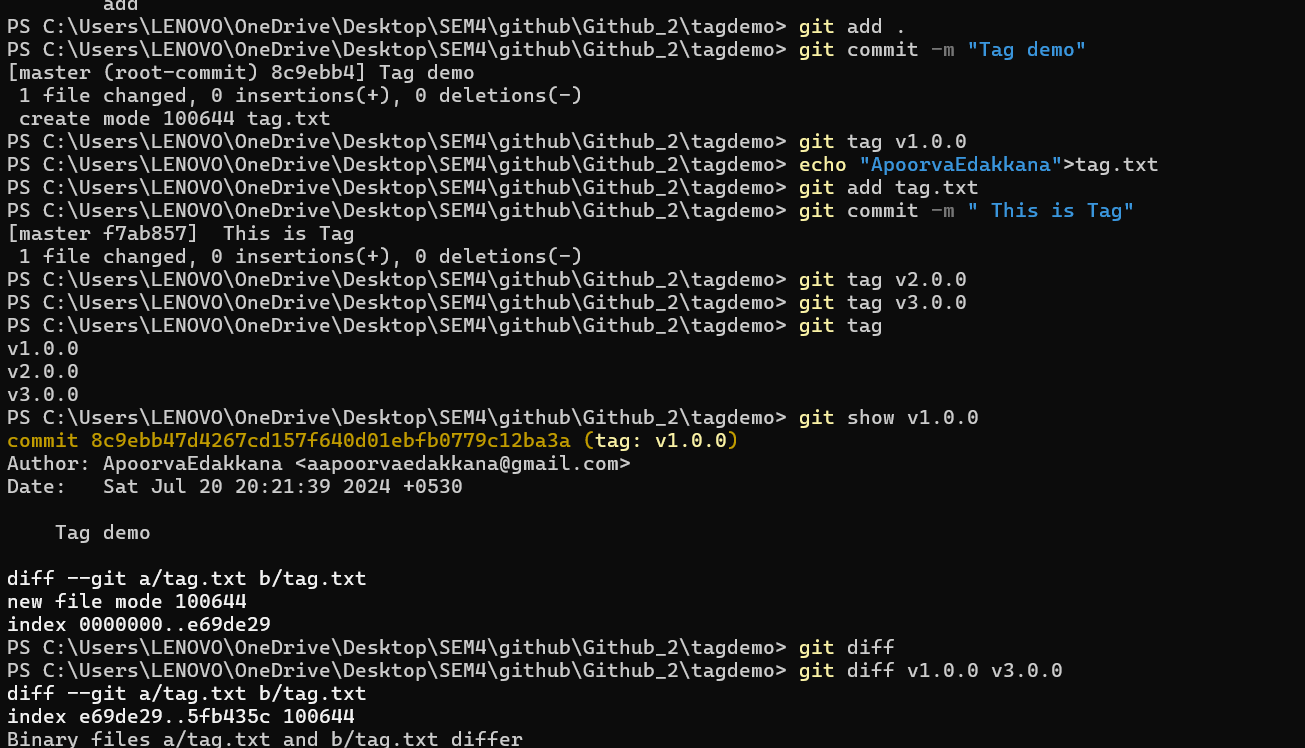
**4.5**.**Procedure and Results**

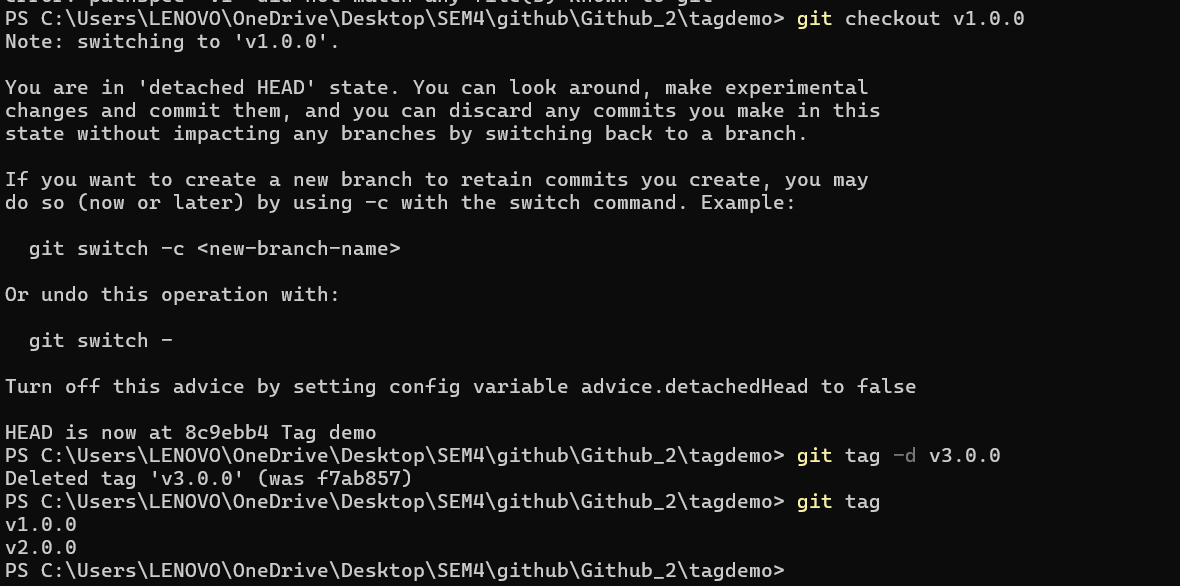
git log

git tag -a v1.0 -m “Tag v1.0”

git show v1.0







# EXPERIMENT No.05: Advanced Git Operations



|  |  |
| --- | --- |
| 5.1 Objective | 5.4 Introduction |
| 5.2 System Configuration | 5.5 Procedure and Results |
| 5.3 Pre-Requisite |  |
|  |  |

**5.1 Objective**

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

**5.2 System Configuration**

Windows 10, Linux, Mac

**5.3 Pre-Requisite**

Install Git bash and make required settings.

**5.4 Introduction**

git checkout -b source-branch:

Creates and switches to a new branch named "source-branch." Makes some change filename: Edits a file in the working directory.

git add . :

Stages all changes in the working directory for the next commit. git push:

Pushes committed changes to the remote repository. git commit -m "commit123":

Commits the staged changes with the given commit message. git status:

Displays the status of changes as untracked, modified, or staged.

git log --oneline:

Shows a concise log of commits with their hash and short descriptions. Copy hash of the commit:

Manually record the commit hash for reference. git checkout -m main:

Switches to the "main" branch, assuming it already exists. git cherry-pick cb0249:

Applies the changes from a specific commit (cb0249) to the current branch. git cherry-pick 5ei3ei...2b:

Picks a range of commits and applies them to the current branch. git cherry-pick --continue:

Continues the cherry-pick process after resolving conflicts. git log --oneline:

Displays an updated log after cherry-picking, showing the new commit(s).

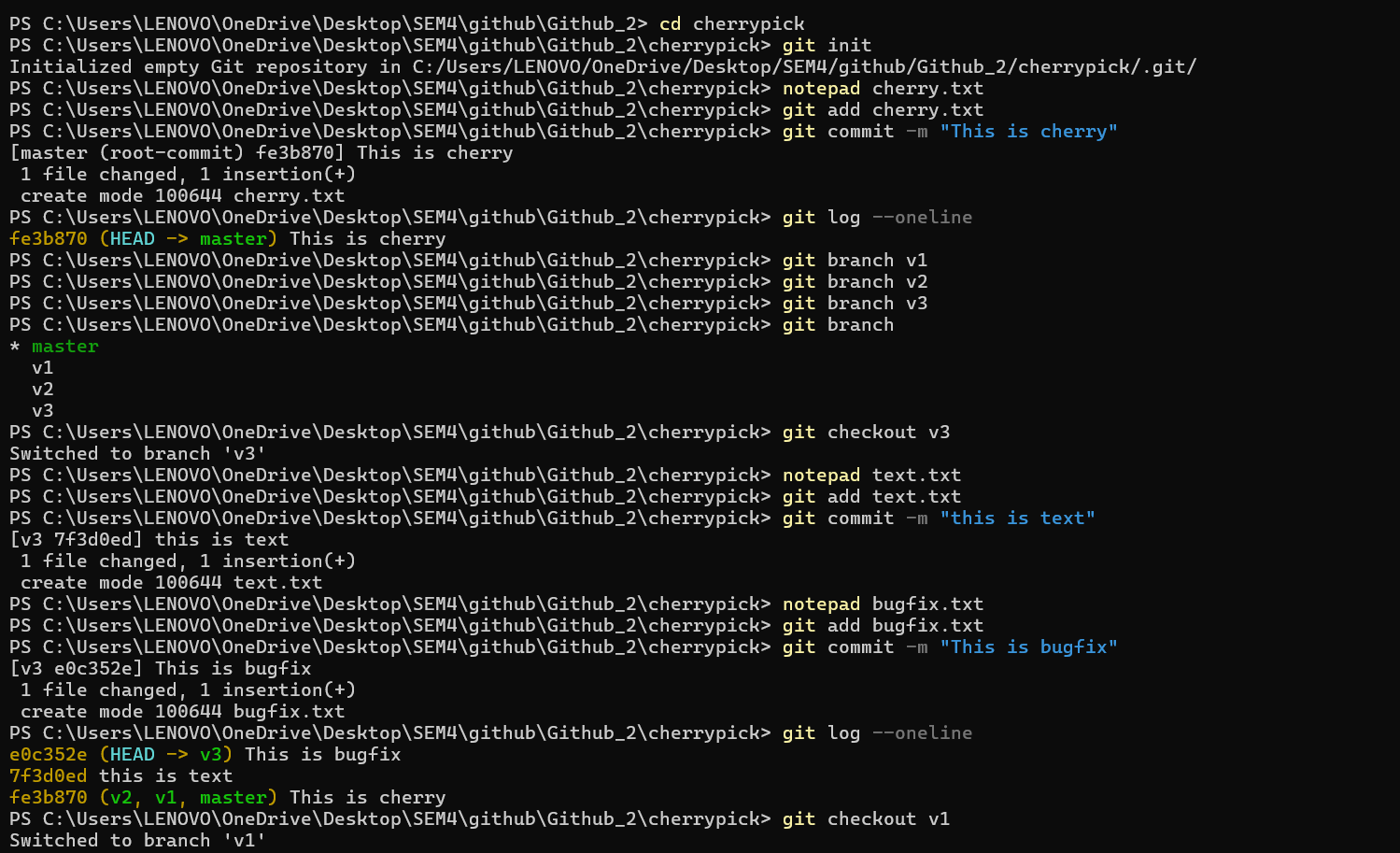
**5.5 Procedure and Results**

git status

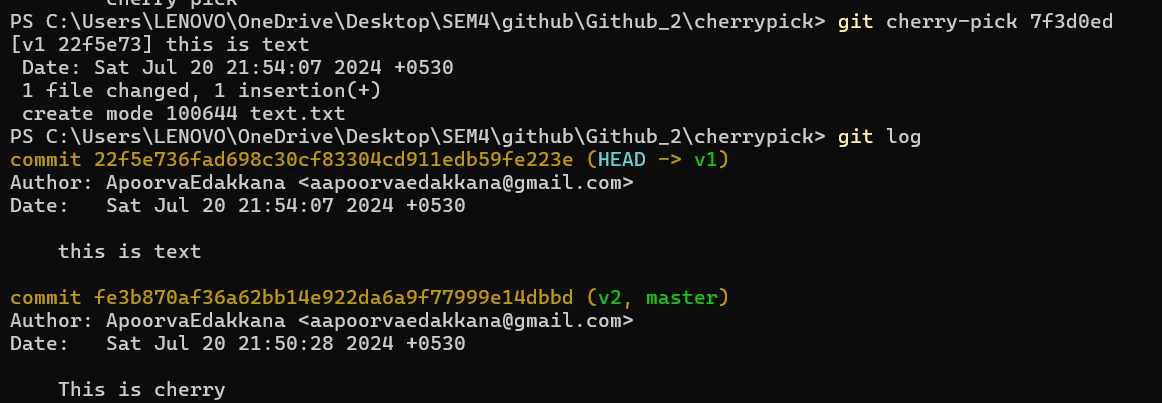
git log --oneline

git checkout feature-branch git cherry-pick <hash>









**EXPERIMENT No.06:** Analzing and Changing Git History



|  |  |
| --- | --- |
| 6.1 Objective | 6.4 Introduction |
| 6.2 System Configuration | 6.5 Procedure and Results |
| 6.3 Pre-Requisite |  |
|  |  |

**6.1 Objective**

1. Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?
2. Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."
3. Write the command to display the last five commits in the repository's history.
4. Write the command to undo the changes introduced by the commit with the ID "abc123".

**6.2 Software Required**

Windows 10, Linux, Mac

**6.3 Pre-Requisite**

Install Git bash and make required settings.

**6.4 Introduction**

git show <commit\_id>

This command displays the details of a specific commit, including the commit message, author, date, and the changes made in the commit. It's useful for reviewing the details of a particular commit in your Git history.

git log --author=JohnDoe --after=2023-01-01 --before=2023-12-31

This command lists all commits made by the author "JohnDoe" between the dates "2023-01-01" and "2023-12-31". It's helpful for filtering the commit history based on authorship and date ranges.

git log -n 5

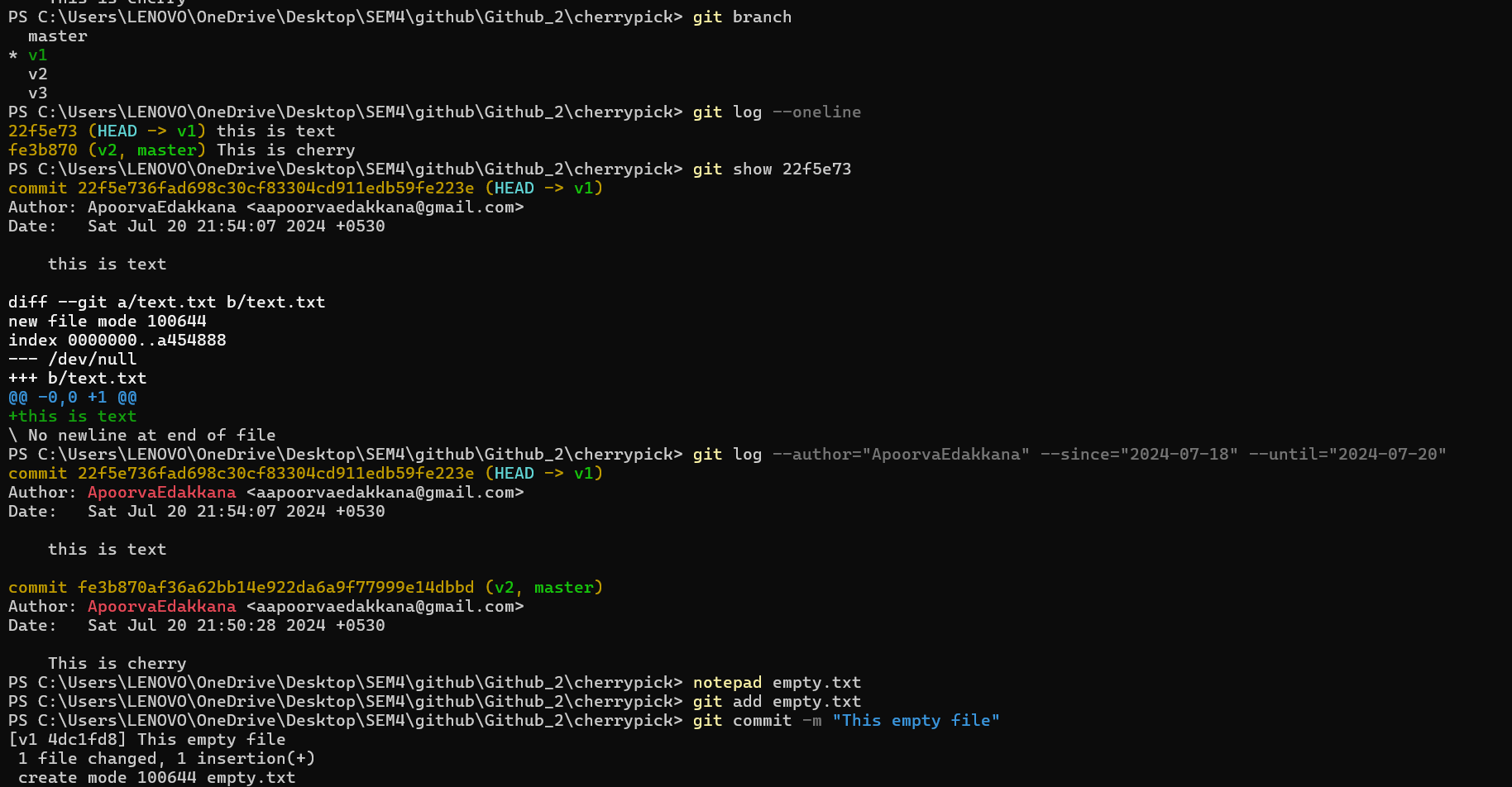
This command displays the last five commits in the repository's history. It's useful for quickly viewing recent changes and understanding the recent development activity in the repository. git revert abc123

This command creates a new commit that undoes the changes introduced by the commit

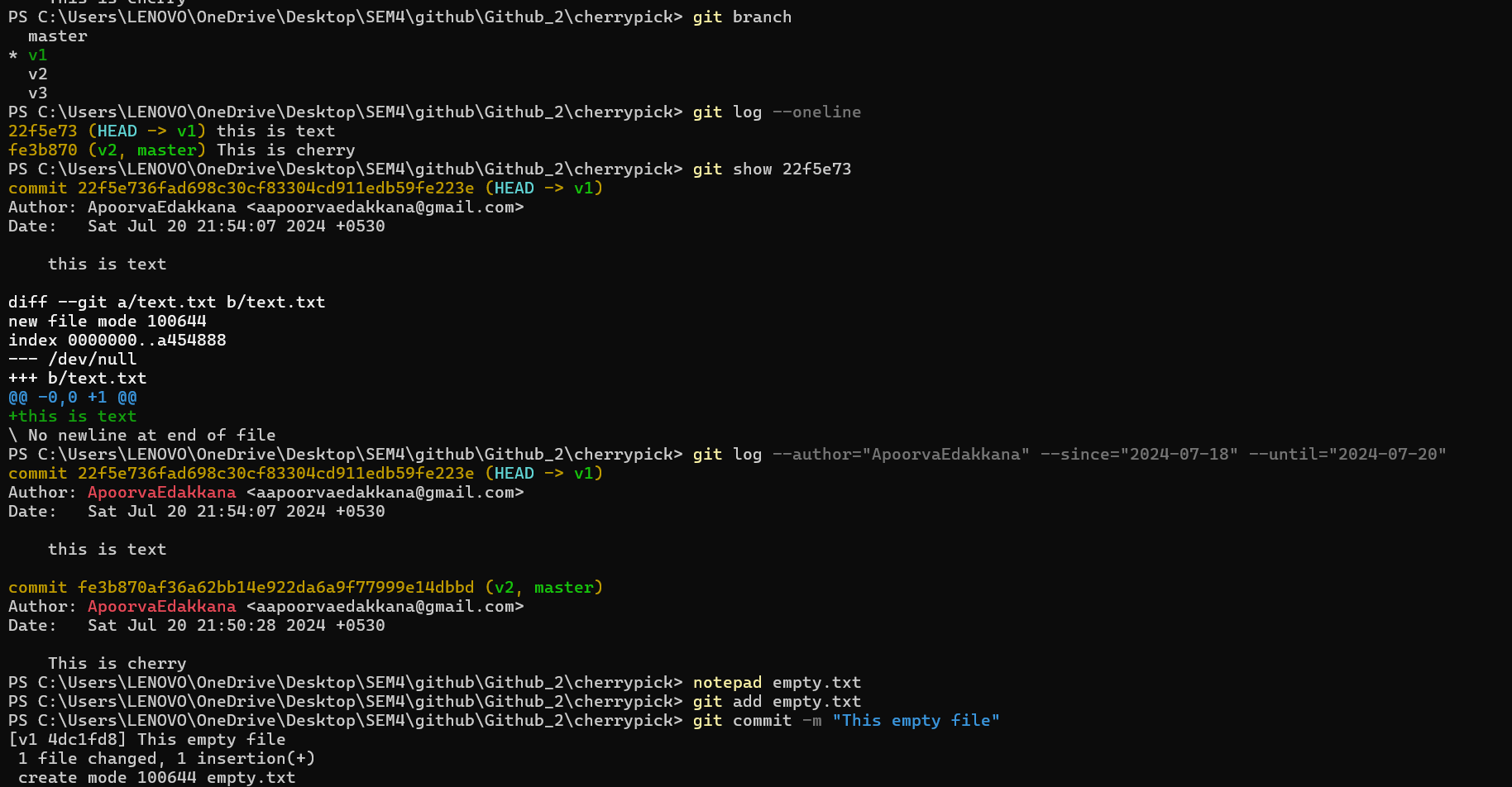
with the ID "abc123". It's a safe way to undo changes without altering the commit history, as it creates a new commit that reflects the changes being reverted.

**6.5 Procedure and Results**

git show <commit\_id>



git log --author=”JohnDoe” --after=”2023-01-01” --before=”2023-12-31”



git log -n 5

