**EXPERIMENT-01**

**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.**

**PROCEDURE**:

1. Initialize a new Git repository:

Open your terminal or command prompt and navigate to the desired directory. Then, use the following command to initialize a new Git repository:

* git init

This command sets up a new Git repository in the current directory.

1. Create a new file:

You can create a new file using a text editor or use the command line. For example, using the command line in Unix/Linux/macOS:

* touch filename.txt

Replace filename.txt with the desired name for your new file.

1. Add the new file to the staging area:

* git add filename.txt

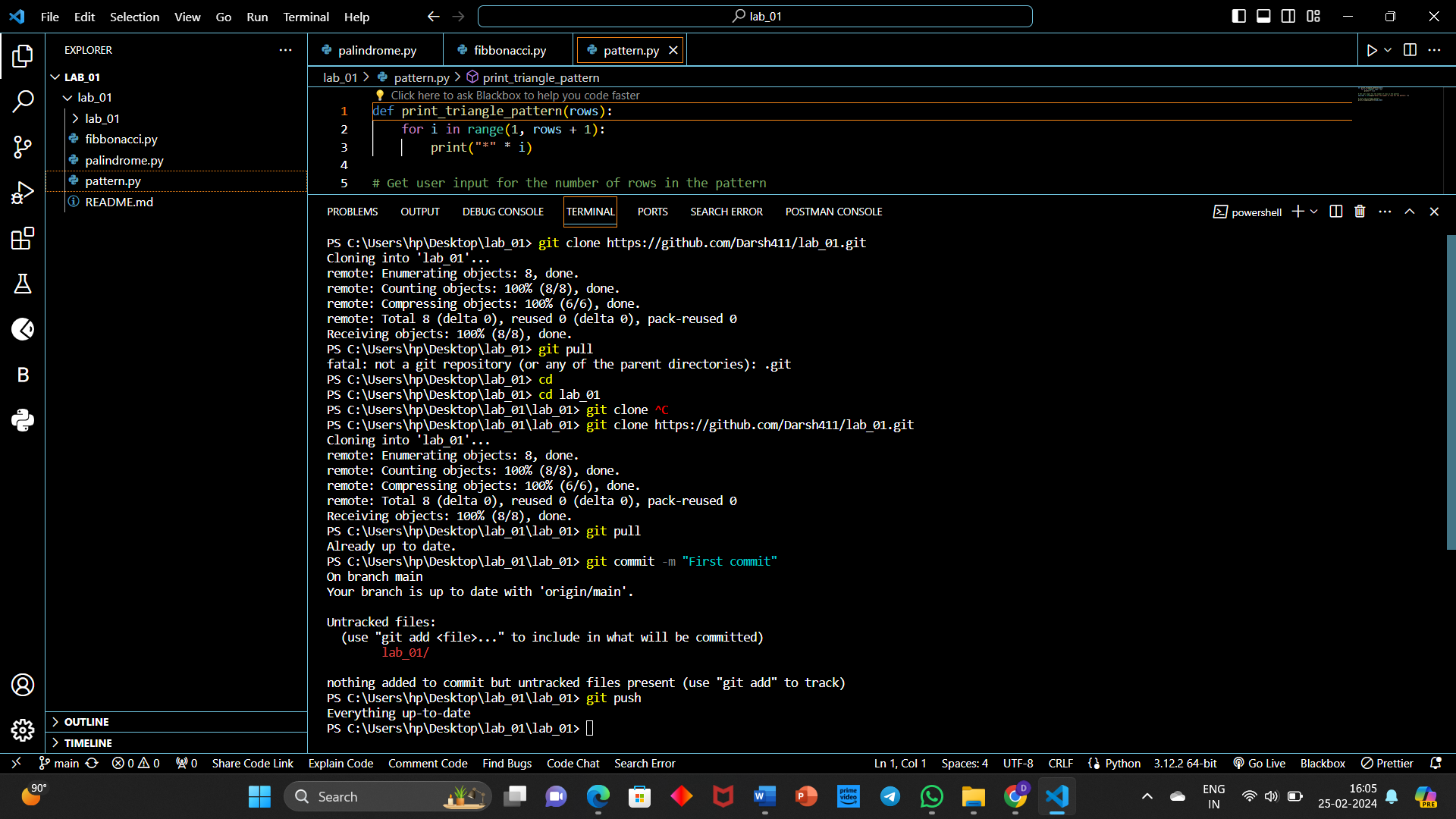
This command stages the changes in the specified file to be committed. You can use git add . to add all files in the current directory.

1. Commit the changes with a message:

* git commit -m "Initial commit"

Replace "Initial commit" with an appropriate commit message describing the changes made in this commit. This command records the changes staged in the previous step to the Git repository.

**OUTPUT**:



**EXPERIMENT-02**

**Creating and Managing Branches**

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.**

**PROCEDURE**:

1. Create a new branch, switch to master, and merge:

Create a new branch named "feature-branch":

* git branch feature-branch

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

1. Switch to the "master" branch:

* git checkout master

or

* git switch master

This command switches to the "master" branch.

1. Merge "feature-branch" into "master":

* git merge feature-branch

This command merges changes from "feature-branch" into the "master" branch.

1. Create a new branch named "feature-branch":

* git branch feature-branch

This command temporarily saves your changes in a stash

2. Switch to the desired branch (e.g., "feature-branch"):

* git checkout feature-branch

or

* git switch feature-branch

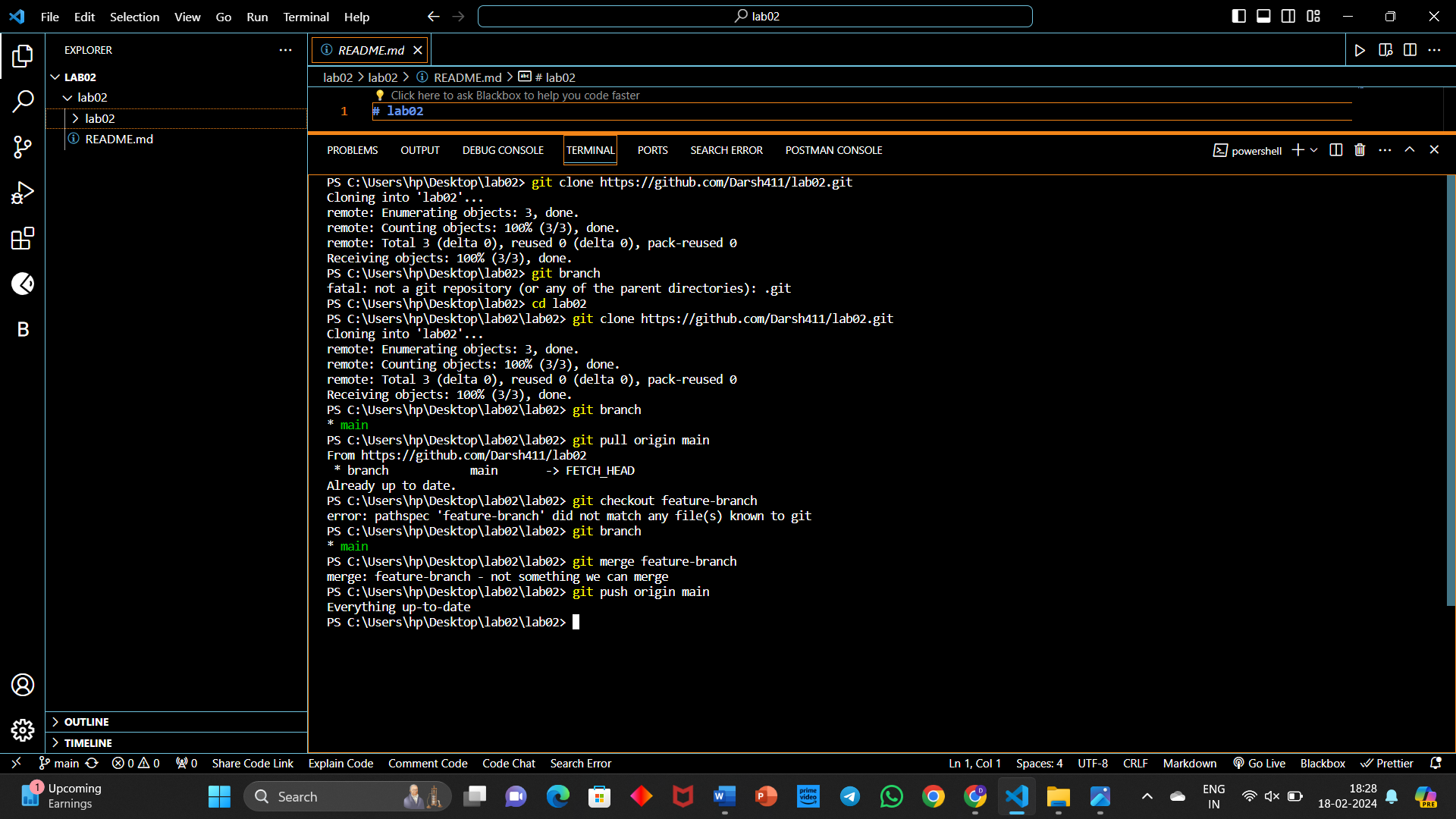
3. Apply stashed changes:

* git stash apply

This command reapplies the most recent stash to your current working directory.

If you have multiple stashes or want to apply a specific stash, you can use git stash apply stash@{n} where n is the stash index.

**OUTPUT**:



**EXPERIMENT-03**

**Collaboration and Remote Repositories**

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

a) Clone a remote Git repository to your local machine:

* git clone <repository\_url>

Replace <repository\_url> with the actual URL of the remote Git repository. This command creates a copy of the remote repository on your local machine.

b) Fetch the latest changes and rebase your local branch:

Assuming you are on your local branch and want to update it with the latest changes from the remote:

* git fetch origin

This command fetches the latest changes from the remote repository.

* git rebase origin/<branch\_name>

Replace <branch\_name> with the name of the remote branch you want to rebase onto. This command incorporates the latest changes from the remote branch into your local branch.

c) Merge "feature-branch" into "master" with a custom commit message:

* git checkout master

Switch to the "master" branch.

git merge feature-branch -m "Merge feature-branch: Implement new feature"

Replace "Implement new feature" with your desired custom commit message. This command merges changes from "feature-branch" into "master" with the specified commit message.

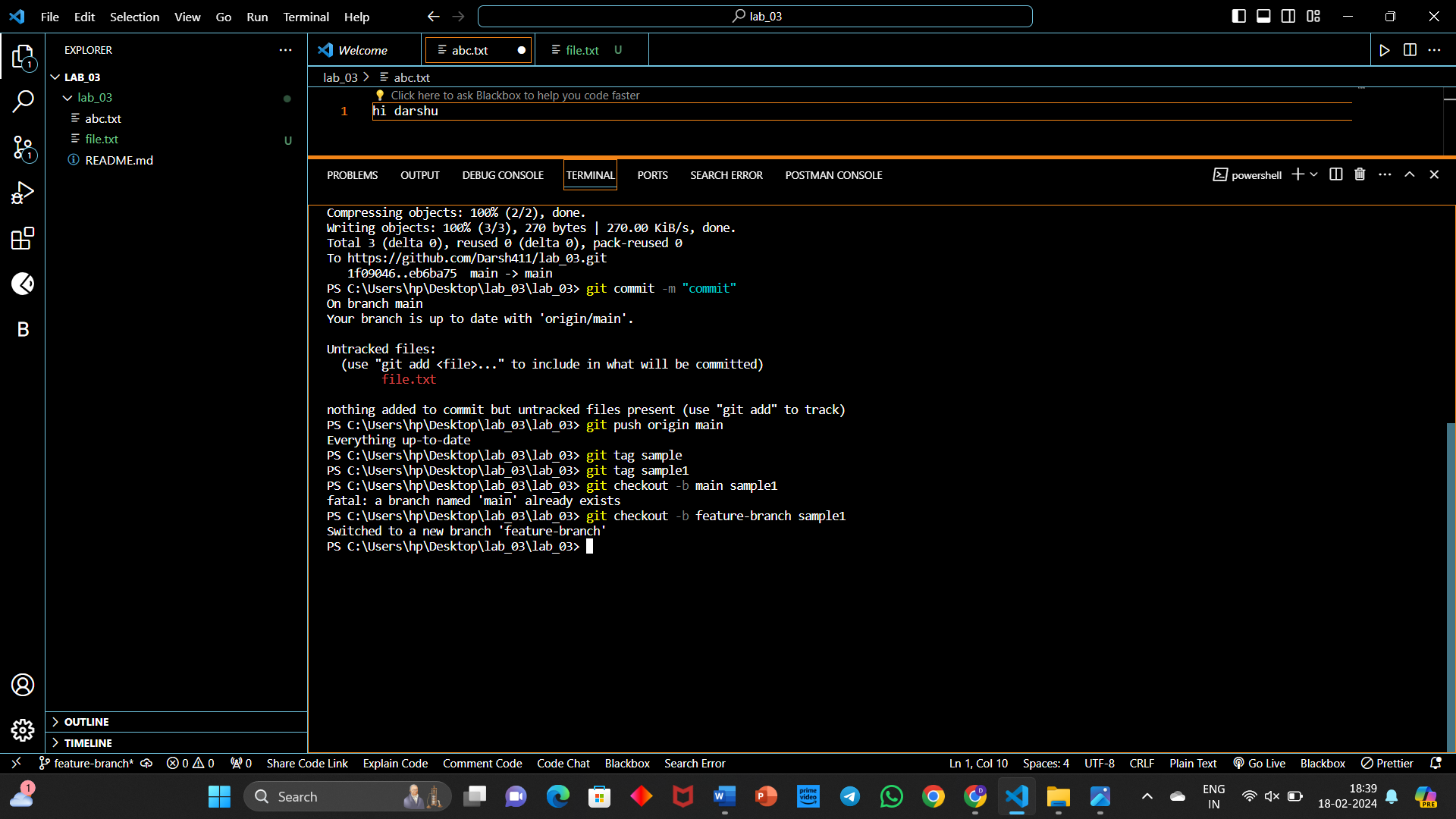
Alternatively, if you prefer to keep a linear history (no merge commits), you can use rebase instead of merge:

* git checkout master

Switch to the "master" branch.

* git rebase feature-branch

This rebases "master" onto "feature-branch" and allows you to resolve any conflicts. After resolving conflicts, you can continue with the rebase. Remember that rebasing rewrites commit history, so use it carefully, especially if the branch is shared with others.

**OUTPUT**:

**EXPERIMENT-04**

**Git Tags and Releases**

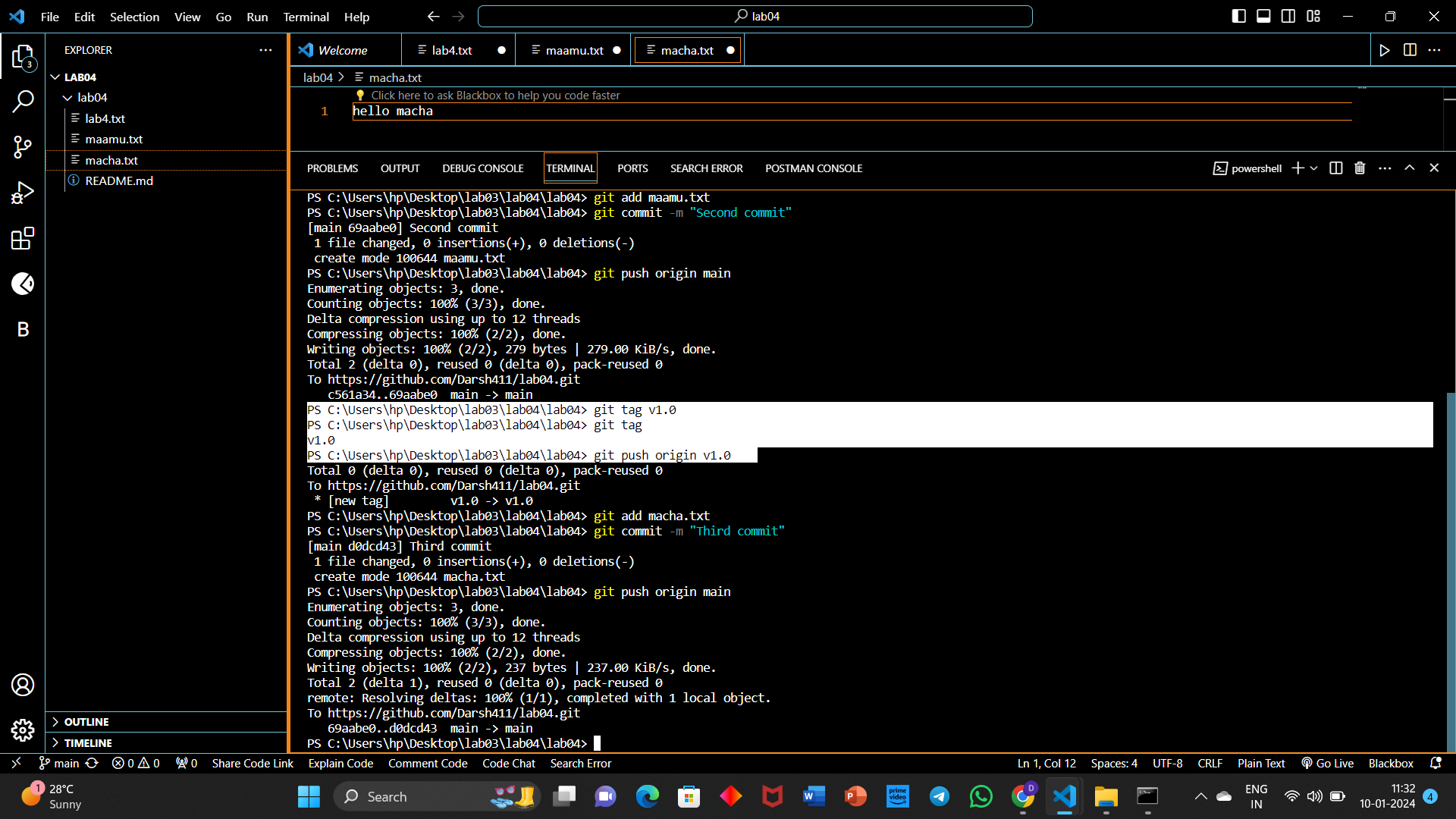
**Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.**

**Git Tags:** It refers to creating specific points in the history for the user’s repository. And the repository is created to make release points for the user’s code.

**PROCEDURE**:

1. Create a repository in git.
2. git clone <url>
3. git add <filename>
4. git push origin <branch>
5. git tag <tag\_name>
6. git checkout -b <branch\_name><tag\_name>
7. Create file1.txt
8. cd clone <link>
9. cd lab04
10. git add file1.txt.
11. git commit -m “First commit”
12. git push origin main
13. Create file2.txt.
14. Repeat the steps (step 8-step 12).
15. git tag v1.0
16. git tag
17. Create file3.txt.
18. Repeat the steps (step 8-step 12).
19. git checkout -b version v1.0
20. git push origin version1

**OUTPUT**:



**EXPERIMENT-05**

**Advanced Git Operations**

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

Cherry-picking in GitHub refers to the process of selecting specific commits from one branch and applying them to another branch. This allows you to pick and choose specific changes or features from one branch and merge them into another, without merging the entire branch.

**PROCEDURE:**

1. Create a New Repository and clone it:

* git clone <url>

2. Create and commit a file in Main Branch:

* echo "File A - Main Branch" > file\_a.txt
* git add file\_a.txt
* git commit -m "Initial commit on main branch"
* git push origin main

3. Create the Source Branch and commit two files in it:

* git checkout -b <source\_branch>
* git push origin source\_branch

4. Switch back to main branch:

* git checkout main

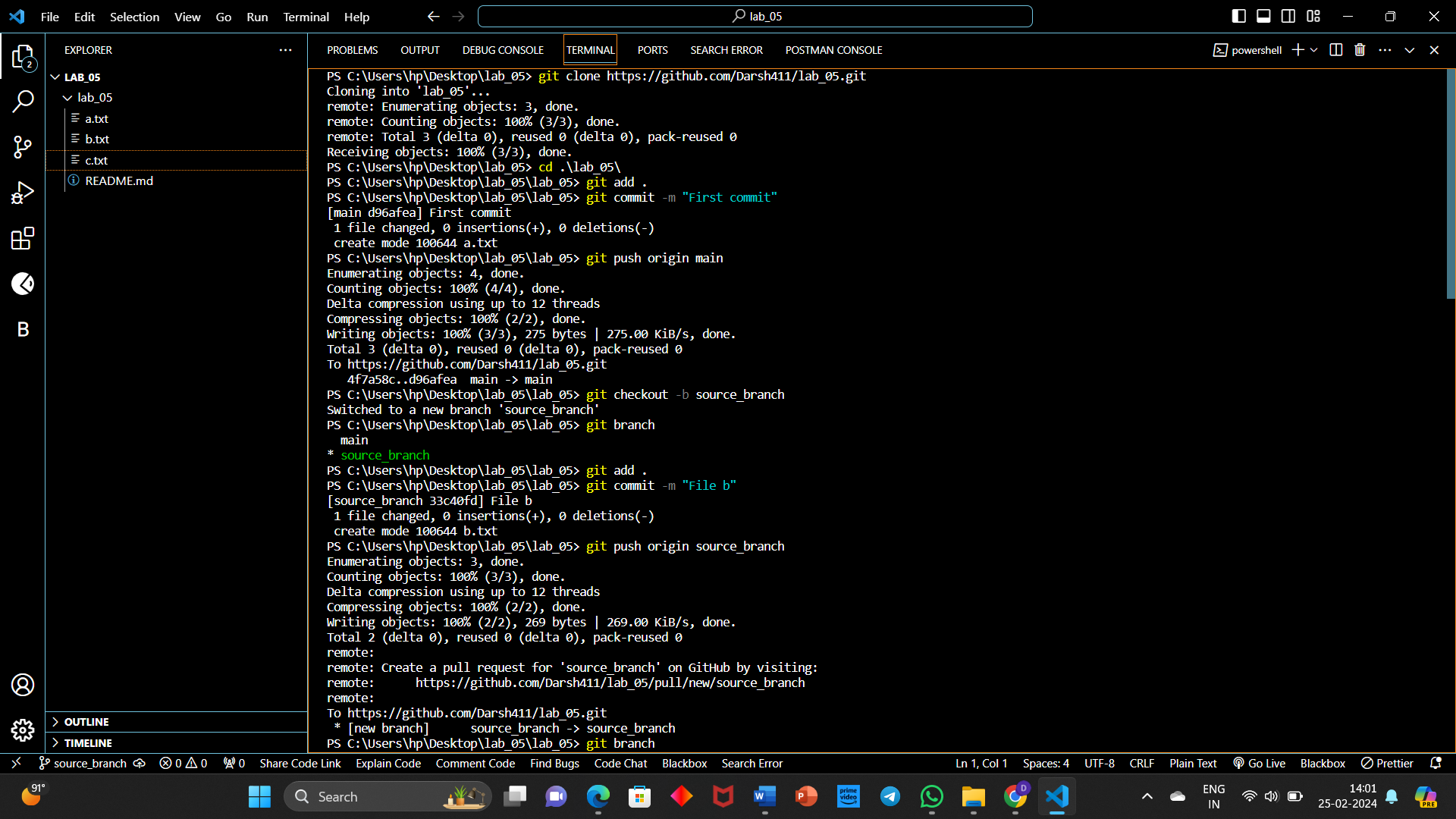
5. cherry pick range of commits from source\_branch to current branch (main):

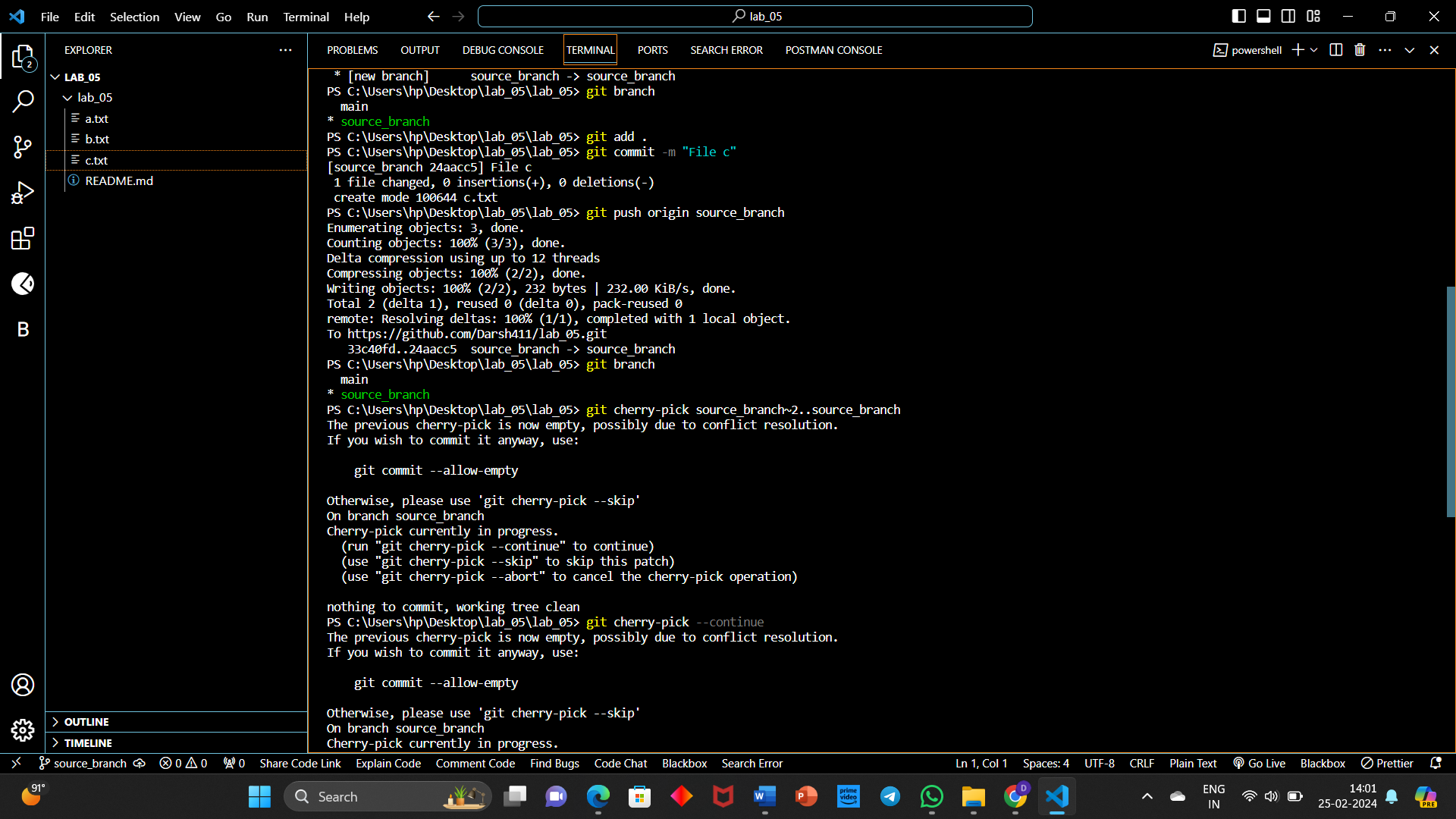
* git cherry-pick source\_branch~<range>..source\_branch

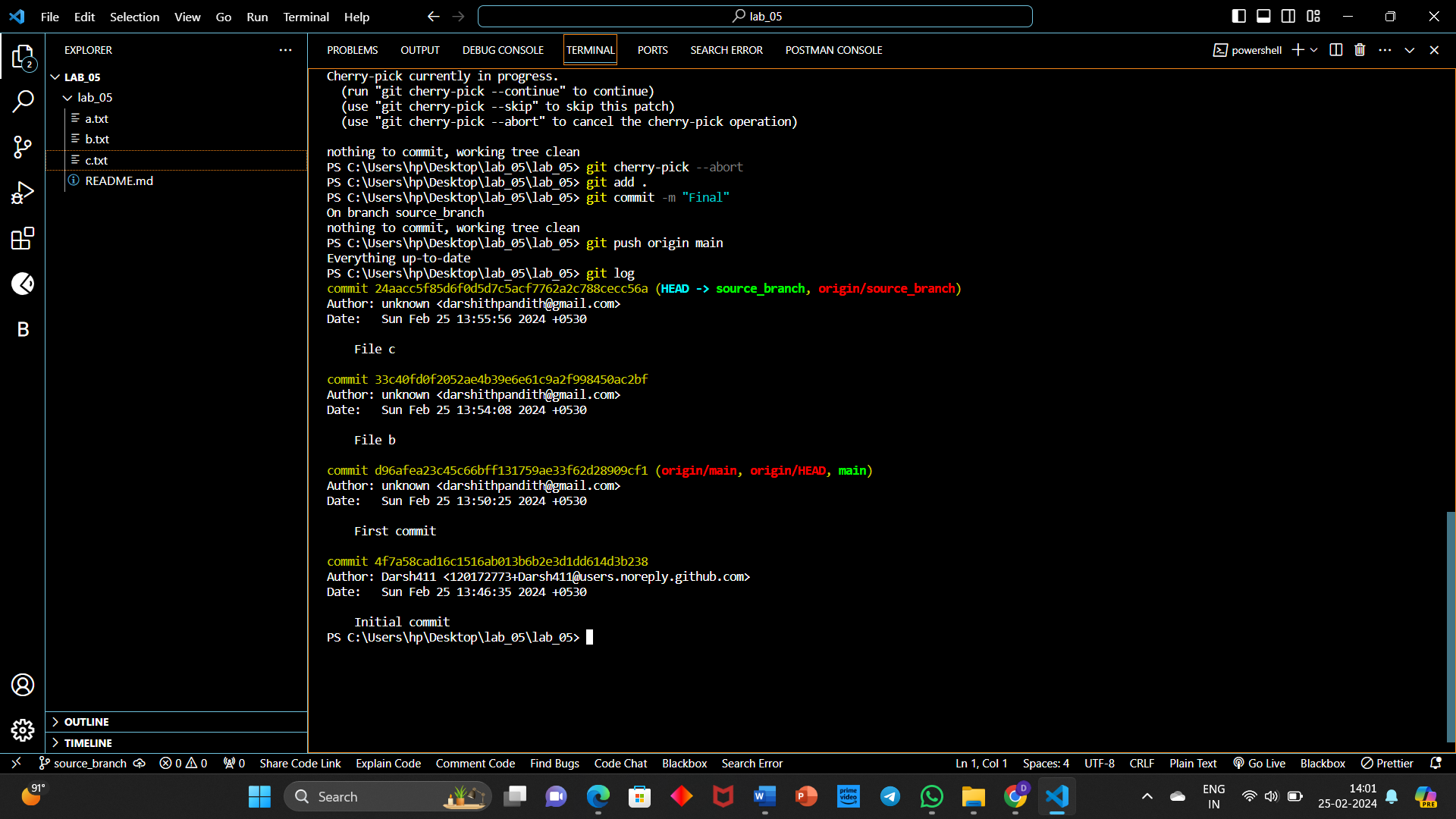
6. Review the changes and push:

* git log
* git push origin main

**OUTPUT**:

****

****



**EXPERIMENT-06**

**Analysing and Changing Git History**

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".**

**PROCEDURE**:

1. Create a New Repository and clone it:

* git clone <url>

2. Create and commit five files in Main Branch:

* echo "File A - Main Branch" > file\_a.txt
* git add file\_a.txt
* git commit -m "Initial commit on main branch"
* git push origin main

3. To list all commits:

* git log

4. To view the details of a specific commit in Git:

* git show <commit\_id>

Replace <commit\_id> with the actual commit ID you want to view.

5. To list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31"

* git log --author="JohnDoe" --since="2024-02-21" --until="2024-02-22"

6. To display the last three commits in the repository's history:

* git log -n 3

7. To undo the changes introduced by the commit with the ID "abc123”:

* git revert abc123

**OUTPUT**:

