

**AMITY INSTITUTE OF INFORMATION AND TECHNOLOGY**

Course Code: CSE

**“SOURCE CODE MANAGEMENT”**

LABORATORY RECORD

Winter 2nd Semester

2024-25

**SUBMITTED TO:**

Dr. Monit Kapoor

Pro Vice Chancellor & Director - ASET

Amity University, Bengaluru, Karnataka.

**SUBMITTED BY:**

Dhruvadhar S (A86605224271)

**INDEX**

Content :

## Setting up Git Environment and Basic Configuration

1. Creating and managing local repository.
2. Working with basic git commands(add,commit,status)
3. Managing Files and Directories in Git
4. Understanding Git Workflow and Staging Area
5. Creating and Switching Branches
6. Merging branches and resolving conflicts
7. Working with Remote Repositories
8. Collaborating using Pull Requests

## **Lab Exercise 1**: Setting up Git Environment and Basic Configuration

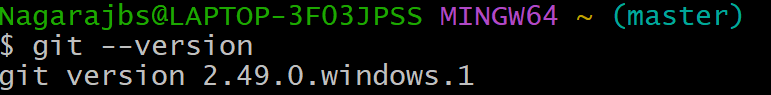
**Objective:**  To install Git version control system and configure basic user settings for version control operations.

**Prerequisites:**  
  Operating System (Windows/Linux/MacOS)  
  Administrator/sudo privileges  
  Internet connection for installation

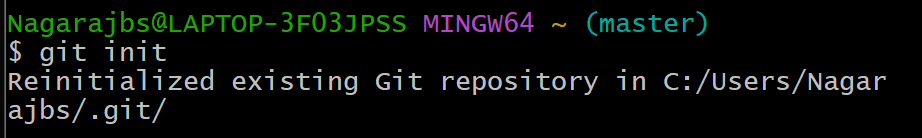
**Environment Details:**  
  Operating System: [OS Build 26100.3624)  
  Git Version: [git version 2.49.0.windows.1]

**Implementation Steps:**  **1. Git Installation:**  
     # Steps used for installation

* Step 1: Download the Git Installer Head over to the official Git website and download the Windows installer.
* Step 2: Run the Installer Open the downloaded setup file to launch the installer.
* Step 3: Select Installation Preferences Choose your desired options as the wizard prompts you.
* Step 4: Complete the Installation Finish the setup by clicking ‘Install.’
* Step 5: Verify the Installation.

**2. Installation Verification:**     # Command – [git --version]  
     # Output – [Installation Verification: git version 2.49.0.windows.1]  
    

**3. Project Setup:**  
     # Commands – [git init]  
     # Output – Reinitialized existing Git repository in C:/Users/Nagarajbs/.git/]



**4. Git Configuration:**     # **Commands** – git config --global user.name”[Nishars021]”

git config --global user.email "[Nisha.s@s.amity.edu]"

# **Verification Commands and Outputs**

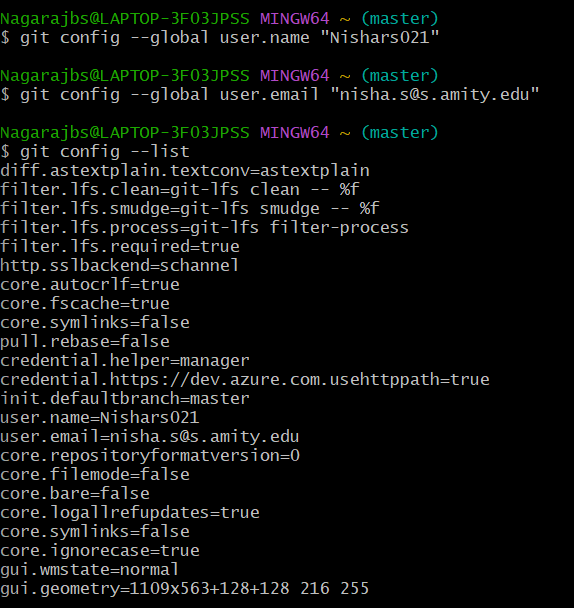
git config --global user.name

git config --global user.email

Expected Output:

user.name=Nishars021

user.email=nisha.s@s.amity.edu



**Challenges Faced:**

 [ sometimes the version of git is not showing in my friend’s laptop it is not displaying the version of the git instead of that it displaying “command not found” ]

**Learning Outcomes:**  1. Understanding of Git installation process  
  2. Knowledge of basic Git configuration  
  3. Ability to verify Git setup and configuration  
  4. Familiarity with Git command-line interface

**Conclusion:**  
  Configured Git for streamlined source code management, for ensuring efficient tracking, collaboration, and version control in development projects.

## **LAB EXERCISE 2:** Creating and managing local repository.

**Objectives:**

To create a local repository and understand its basic structure and initialization process.

**Prerequisites:**

**1**.git

**2.**Basic command line familiarity

**3.**Project directory for familiarity

**Environmental details:**

**1.**Git version: git version 2.49.0.windows.1

**2**.working directory;/c/Users/Nagarajbs

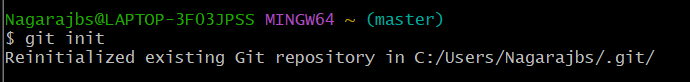
**Implementation steps:**

**1.Repository creation**

**#commands**

git init

**#output**



**2.Repository structural analysis:**

**#COMMANDS**

cd D:

ls-al

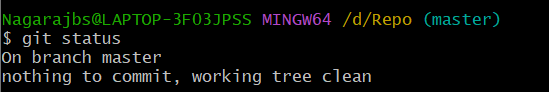
**#output**

**3.Repository status check:**

**#commands**

git status

**#expected output**



Lab exercise 3**:** Working with basic git commands(add,commit,status)

**Objective:**

To understand and practice the basic git commands for staging and committing changes while monitoring repository status

**Prerequisites:**

1.Git repository initialization

2.Basic understanding of git directory structure

**Environment details:**

1.Git version : git version 2.49.0.windows.1

2.Working directory: /source

**Implementation steps :**

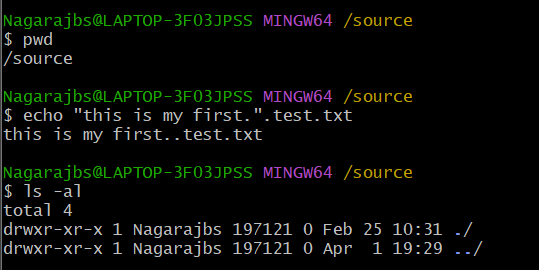
**1.Creating a sample file**

**#commands**

1.echo "this is my first.".test.txt

2.ls -al

**#output**

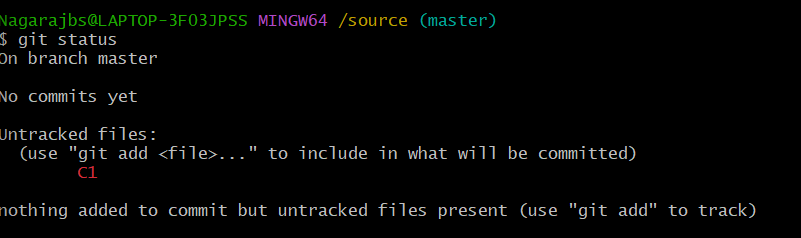


**2.Checking repository status:**

**#commands**

1. git status

**#output**



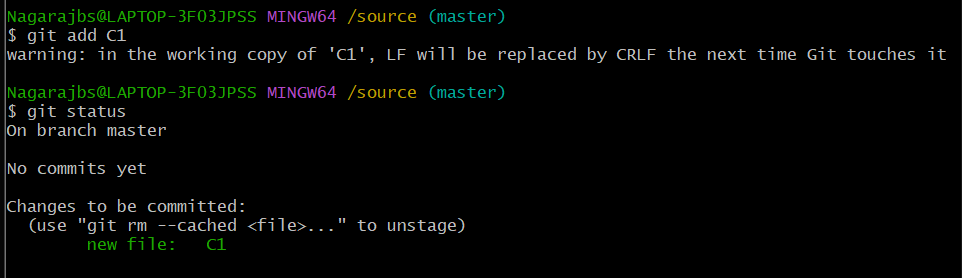
**3.Adding file to staging area:**

**#command**

1.git add C1

2.git status

**#output**



**4.Adding multiple files:**

**#commands**

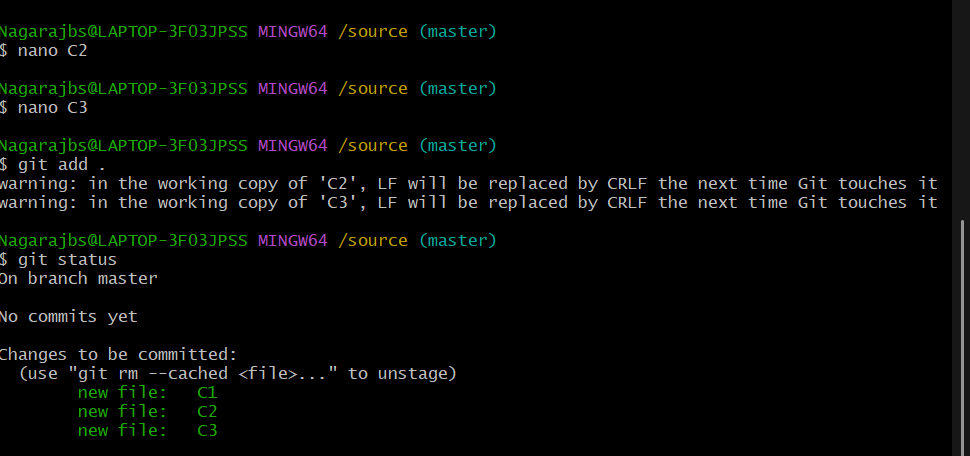
1 nano C2

2. nano C3

3.git add .

4.git status

**#output**

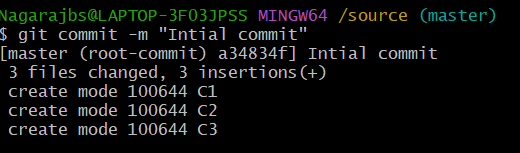
****

**5.Creating initial commit:**

**#commands**

git commit -m “Initial commit”

**#output**



**6.Modifying files:**

**#commands**

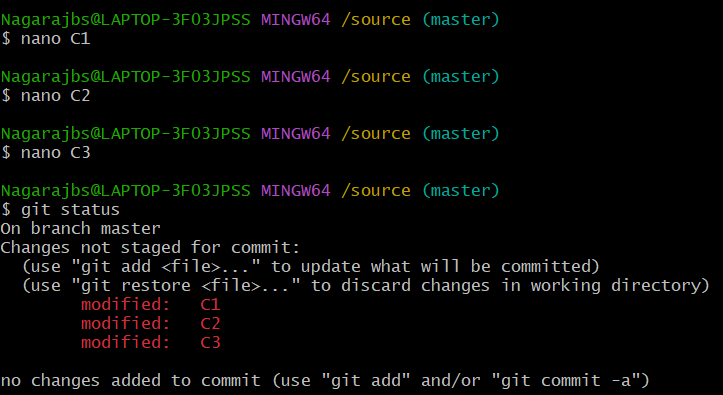
1 nano C1

2. nano C2

3 nano C3

4.git status

**#output**

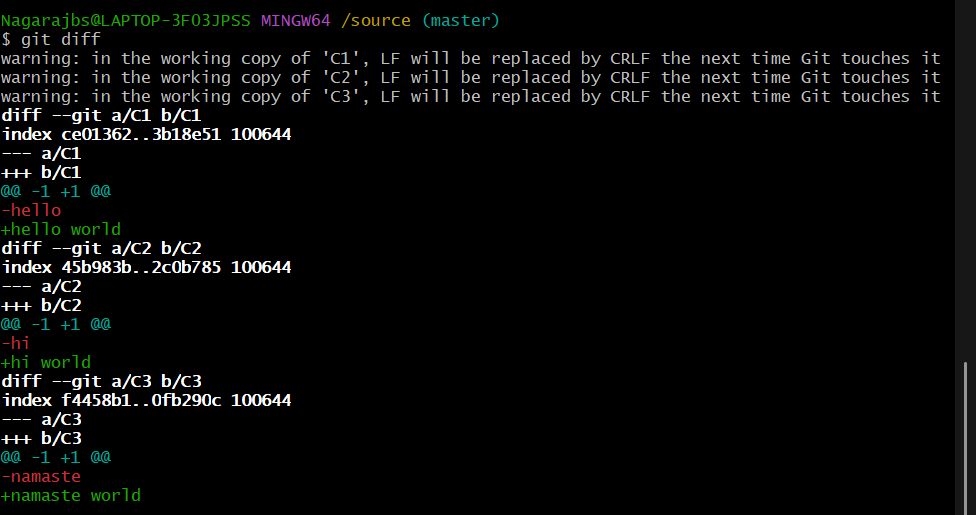


**7.Viewing changes:**

**#commands**

1.git diff

**#output**



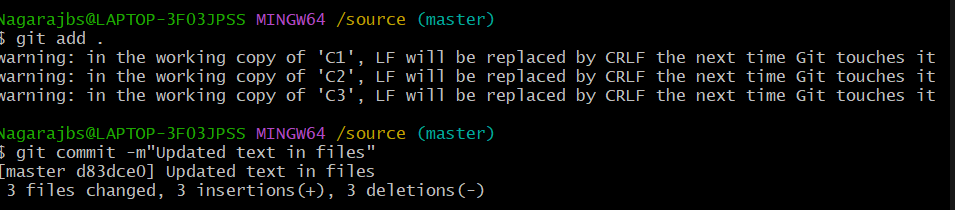
**8.Adding and committing modified files:**

**#commands**

1.git add .

2. git commit -m "Updated text in files"

**#output**



**Challenges faced:**

No challenges faced

**Learning outcomes:**

1.Understanding of staging area concept

2.Mastery of git commands (add,commit,status)

3.Ability to track file changes

4.Knowledge of commit message best practices

5.Understanding difference between tracked and untracked file

**Conclusion:**

Successfully learned and implemented basic git commands for staging and committing changes establishing fundamental version control workflow

## 

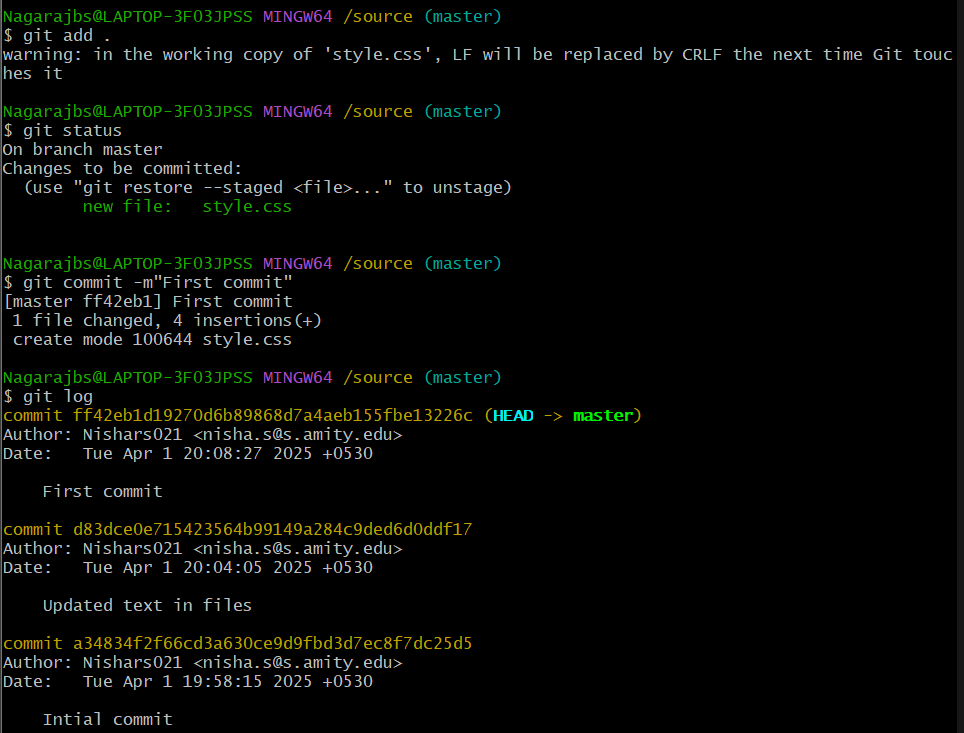
## **Lab Exercise 4:** Managing Files and Directories in Git

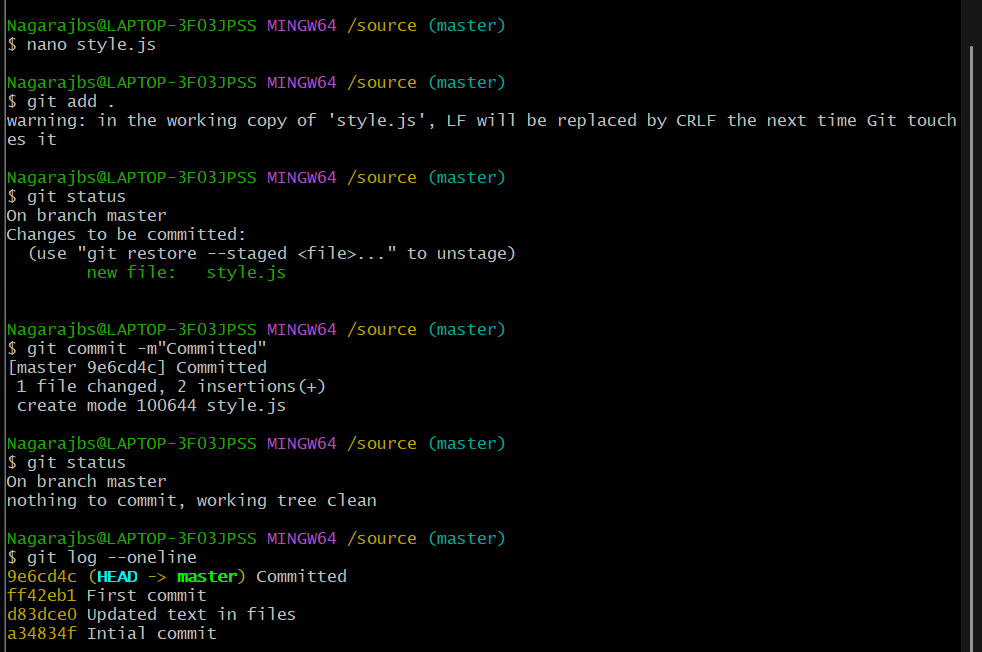
**Objective:**  To understand and practice Git commands for managing files and directories, including adding, tracking modifications, removing, and renaming files.

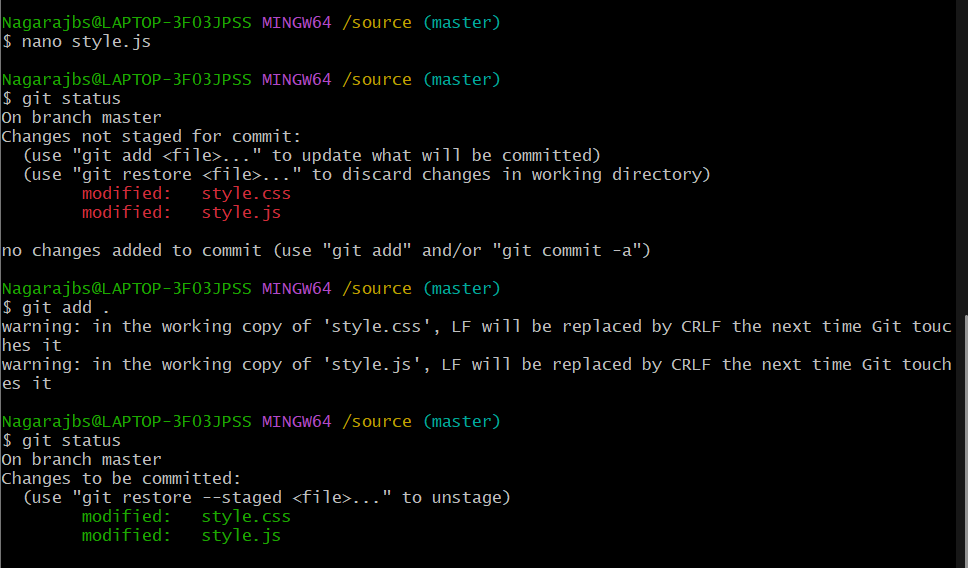
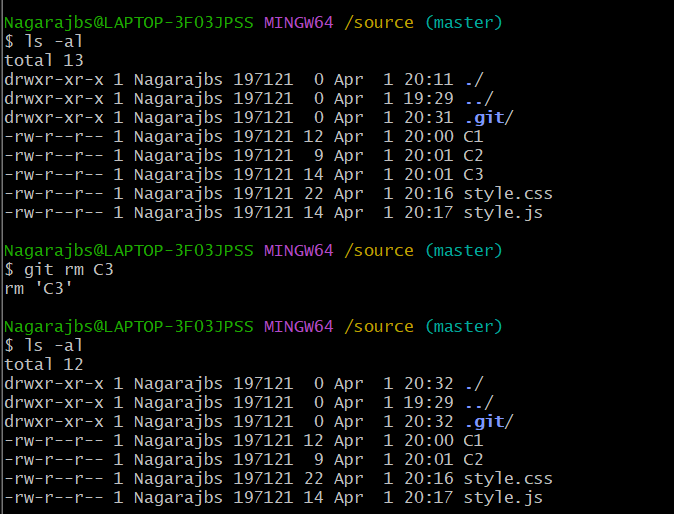
**Prerequisites:**  Git repository initialization  
  Basic understanding of Git commands (add, commit, status)  
  Existing repository with previous commits

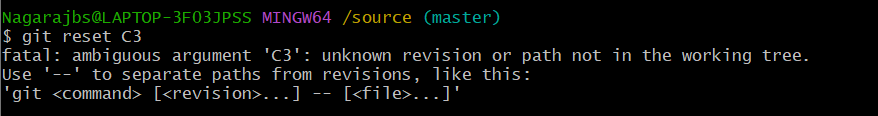
**Environment Details:**  
  Git Version: git version 2.49.0.windows.1  
  Working Directory: /source

**Implementation Steps:**  
  1. Adding New Files to Repository:  
  # Commands:

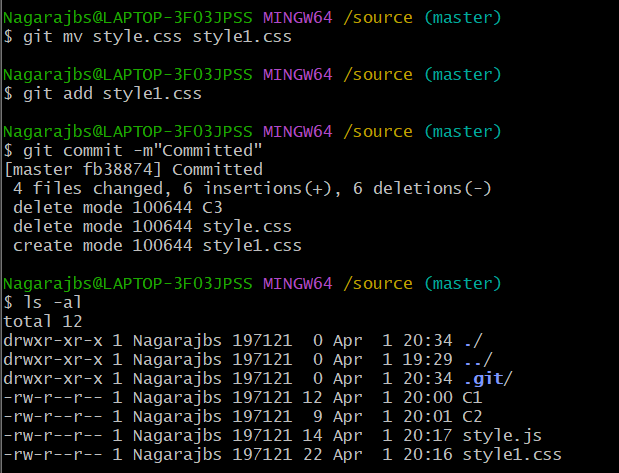
# Output: 

**2. Tracking Modifications:**  
  # Commands : (git diff)  
  # Output :   


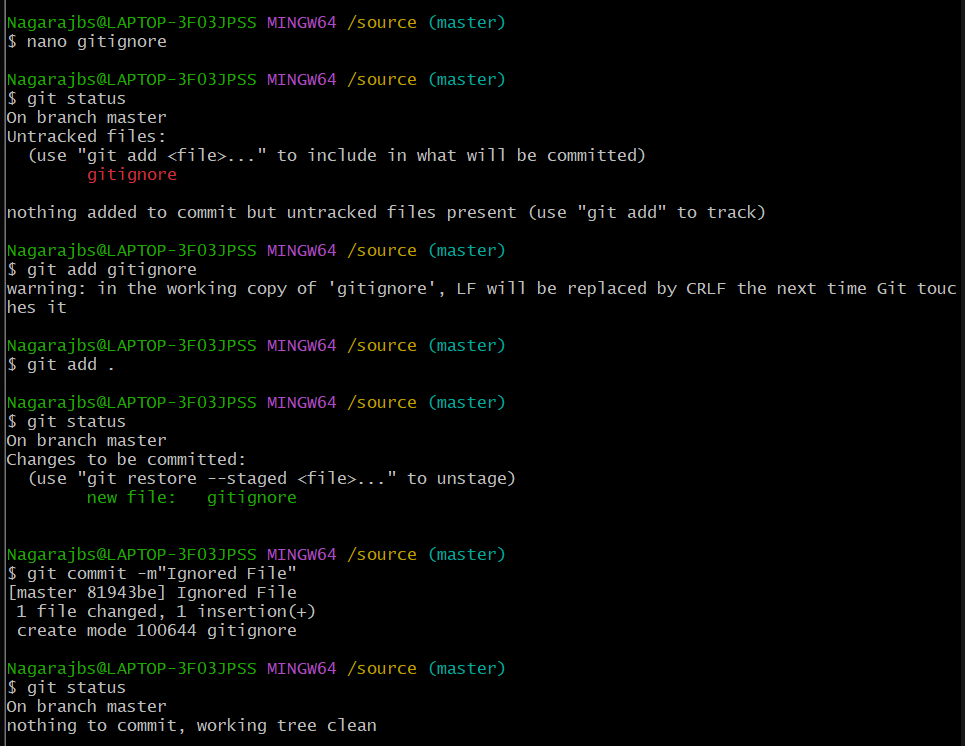
**3. Adding All Changes at Once:**  
  # Commands: git add .  
  # Output  
   **4. Removing Files from Repository**:  
  # Commands: git rm C3  
  # Output  


**5. Removing Files from Staging Area Only**:  
  # Commands : git reset   
  # Output  


**6. Renaming Files in Git:**  # Commands: git mv style.css style1.css  
  # Output



**7. Moving Files to Directories**:  
  # Commands - mv style1.css

**8. Ignoring Files:**  
  # Commands  
  # Output  


**Challenges Faced:**[Document any issues with file management, understanding Git's tracking behavior, or handling file removals and renames]

**Learning Outcomes:**  
1. Understanding of Git's file tracking mechanisms  
2. Ability to add, modify, remove, and rename files using Git commands  
3. Knowledge of efficient file management techniques in Git repositories  
4. Understanding of the .gitignore file for excluding unwanted files  
5. Experience with organizing repository structure while maintaining version history

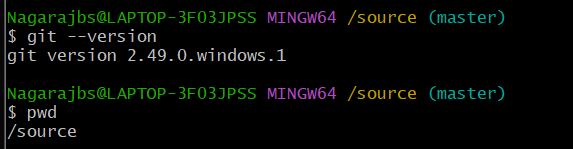
**Conclusion:**  
Successfully learned and implemented Git commands for comprehensive file and directory management, establishing efficient workflows for maintaining repository organization and tracking file changes throughout the development process.

## **Lab Exercise 5:** Understanding Git Workflow and Staging Area

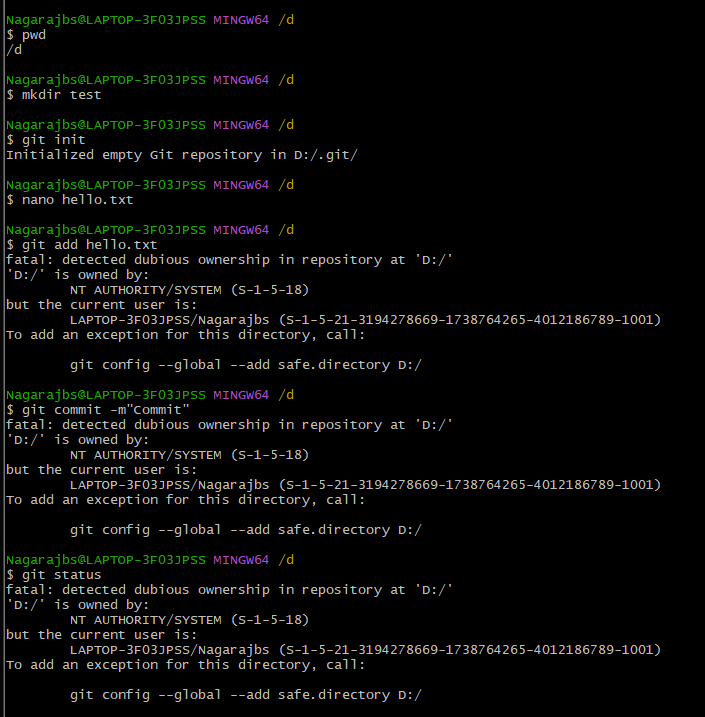
**Objective**:  
To understand and practice Git's three-stage workflow, staging area functionality, and tracking file states throughout the development process.

**Prerequisites:**- Git repository initialization  
- Basic understanding of Git commands (init, add, commit)  
- Command line interface proficiency

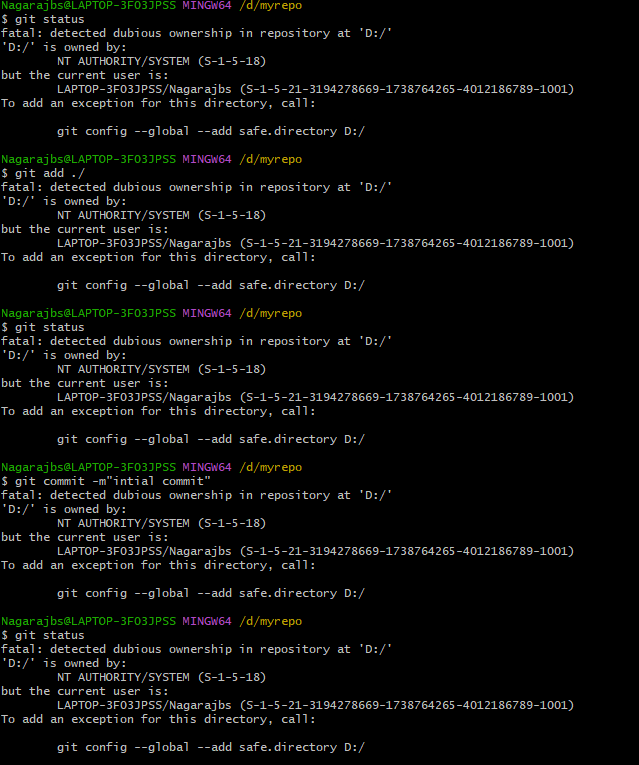
**Environment Details**:  
- Git Version: git version 2.49.0.windows.1   
- Working Directory: /source



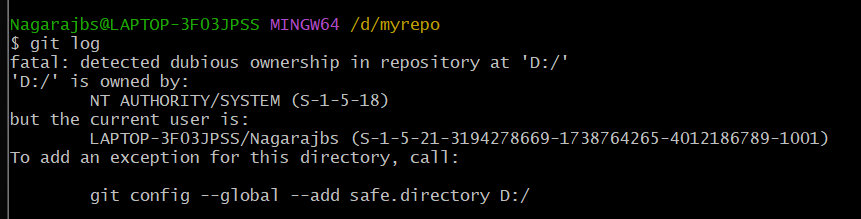
**Implementation Steps:**  **1. Git Data Model Exploration**:  
  #Commands  
  #Output



**2. Three-Stage Workflow Demonstration**:  
  #Commands  
  #Output



**3. Tracking File States**:  
  #Commands  
  #Output



**Challenges Faced:**  
[Document any issues understanding the staging area, or tracking file states]

**Learning Outcomes**:  
1. Understanding of Git's internal data model with commits, trees, and blobs  
2. Practical experience with the three-stage workflow (working directory, staging area, repository)  
3. Ability to track and manipulate file states (untracked, modified, staged, unmodified)  
4. Experience with Git's diff commands to visualize changes between different stages

**Conclusion:**  
Successfully explored and implemented Git's workflow mechanisms, gaining practical understanding of how Git tracks changes through its staging area. Demonstrated the ability to monitor and manipulate file states throughout the development process, creating a foundation for efficient version control practices.

## 

### Lab Exercise 6: Creating and Switching Branches

**Objective:**   
To understand and practice Git's branching system, including creating branches, switching between branches, and managing branch operations.

**Prerequisites:**  
- Git repository initialization  
- At least one commit in the repository  
- Basic understanding of Git commands (init, add, commit)  
- Command line interface proficiency

**Environment Details:**  
- Git Version: git version 2.49.0.windows.1  
- Working Directory: /d/myrepo

**Implementation Steps:**

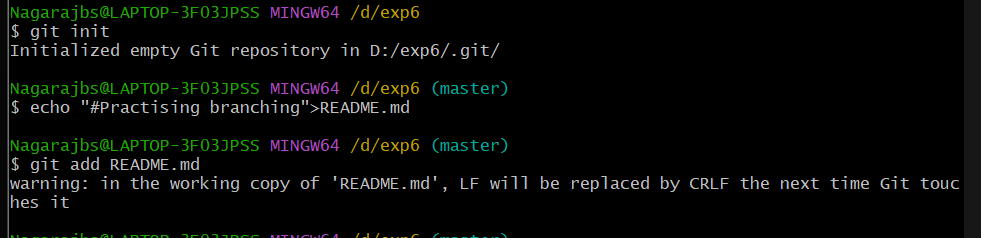
1. **Branch Creation:**  
     **#Commands**

**# Initialize repository -**

git init

echo "#Practising branching">README.md

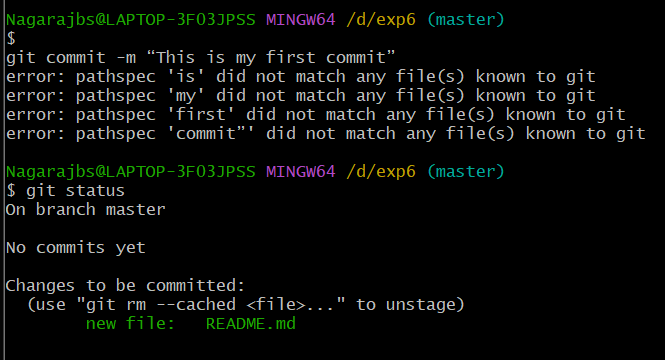
   git add README.md



**# Create initial commit**

git commit -m “This is my first commit”

git status



**# Create new branch**

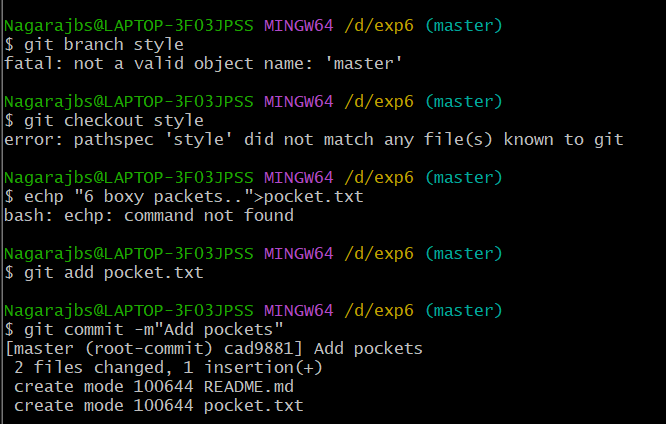
   git branch style

      git checkout style

      echo "6 boxy packets.."> pocket.txt

      git add pocket.txt

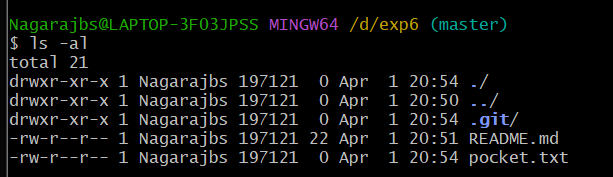
      git commit -m"Add Pocket"

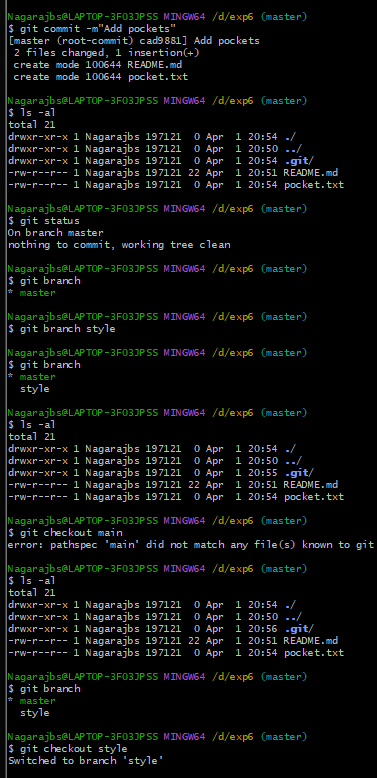


**# Create and switch to a new branch in one command**

[git checkout -b (branch name) fabric]

**#Output**





1. **Switching Between Branches:**  #Commands

git checkout <branchname>   
  # Switch back to main branch

 git checkout master

  # View all branches in repository

 git branch

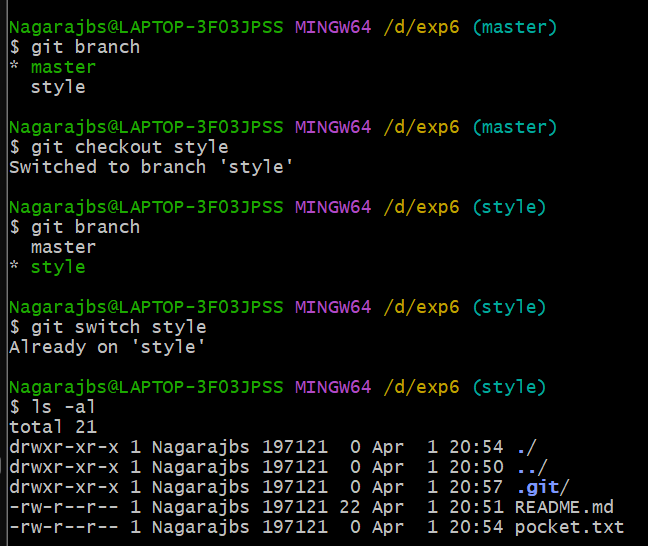
  # Switch to another branch

 git checkout <branchname>

  # Use newer Git switch command

 git switch <branchname>

  #Output



1. **Branch Management:**  #Commands

 git checkout master

  # List all local branches

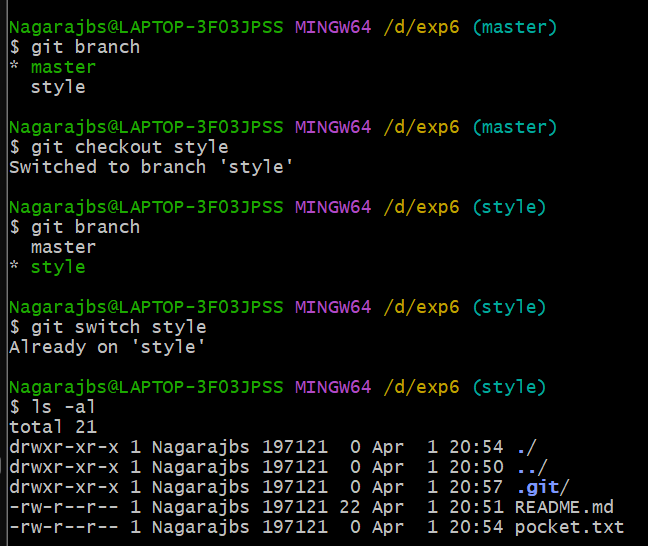
   git branch

  # List all branches (including remote)

  # Delete a branch

git branch -d <branchname>

    git branch -D <branchname>    
  # Rename current branch



**Challenges Faced:**[Document any issues encountered while creating branches, switching between them, or managing branch operations]

**Learning Outcomes:**1. Understanding Git's branching model   
2. Ability to create multiple branches for parallel development work  
3. Experience switching between different branches to isolate development tasks  
4. Practical knowledge of branch management tasks including deletion and renaming

**Conclusion:**  
Successfully explored and implemented Git's branching functionality, gaining practical understanding of how branches enable parallel workflows and feature isolation. Demonstrated the ability to create, navigate between, and manage branches effectively, establishing a foundation for collaborative development practices.

**Lab Exercise 7**: Merging branches and resolving conflicts

Objective:  
To understand and practice two fundamental Git merge strategies: Fast-Forward Merge and Three-Way Merge.

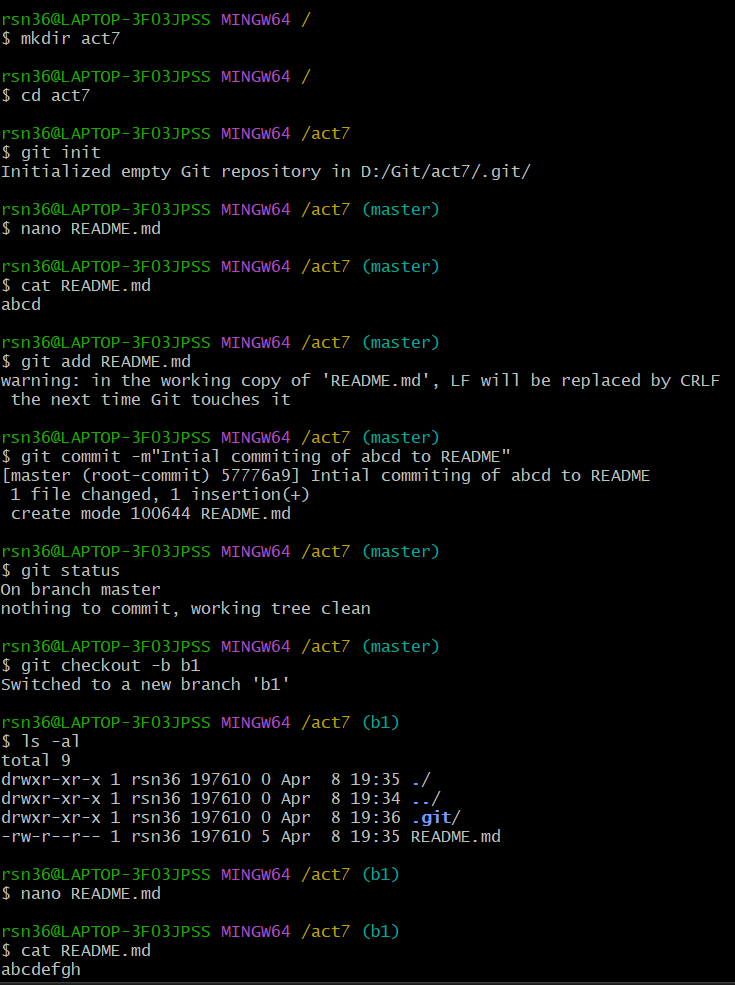
Prerequisites:  
- Git installed  
- Basic understanding of Git branches  
- Command line interface familiarity

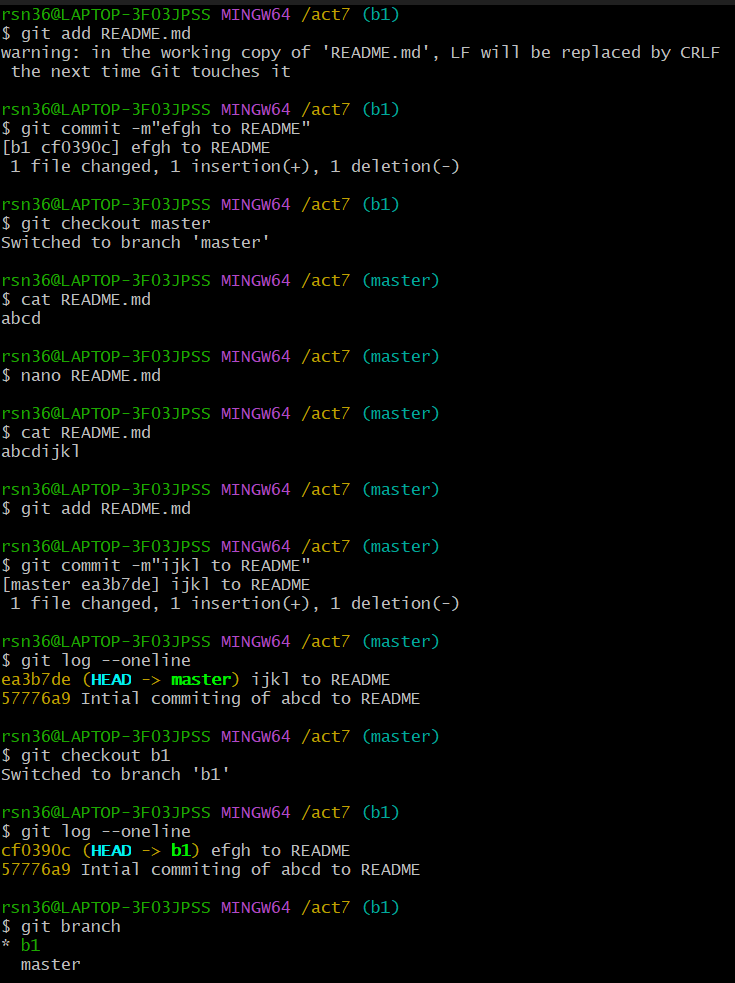
Environment Setup:

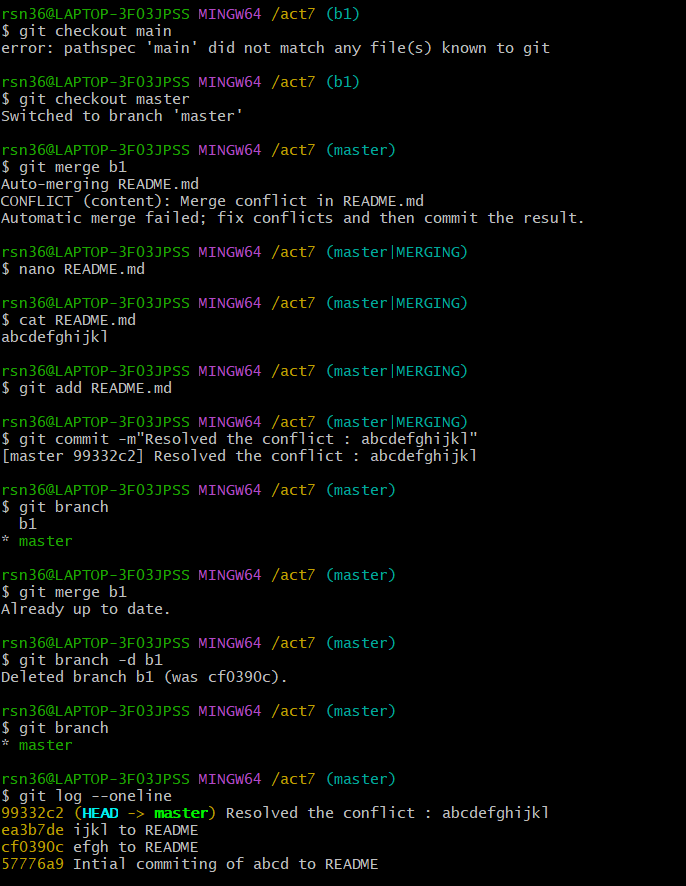
1. Fast-Forward Merge Demonstration:

Scenario:  
Create a repository and demonstrate a simple, linear merge where no additional changes exist in the main branch.

2. Three-Way Merge Demonstration:

Scenario:  
Create a scenario where both main and feature branches have different commits. 





Learning Outcomes:  
1. Understanding different Git merge strategies  
2. Practicing branch merging techniques  
3. Recognizing when different merge approaches are used

Conclusion:  
Merging is a fundamental Git operation that allows developers to combine work from different branches, enabling parallel development and collaborative workflows.

**Lab Exercise 8**: Working with Remote Repositories

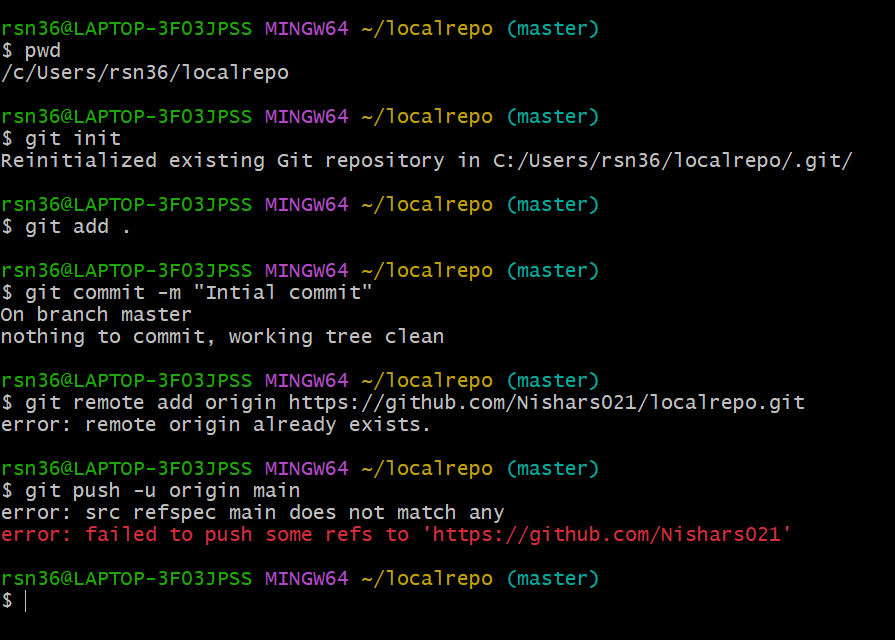
Objective  
To understand and practice working with remote Git repositories, including connecting to remote repositories, pushing local changes, and pulling remote changes.

Prerequisites  
- Git installed  
- Basic understanding of Git branches and commits  
- Command line interface familiarity  
- GitHub/GitLab/Bitbucket account (or any Git hosting service)

Environment Setup

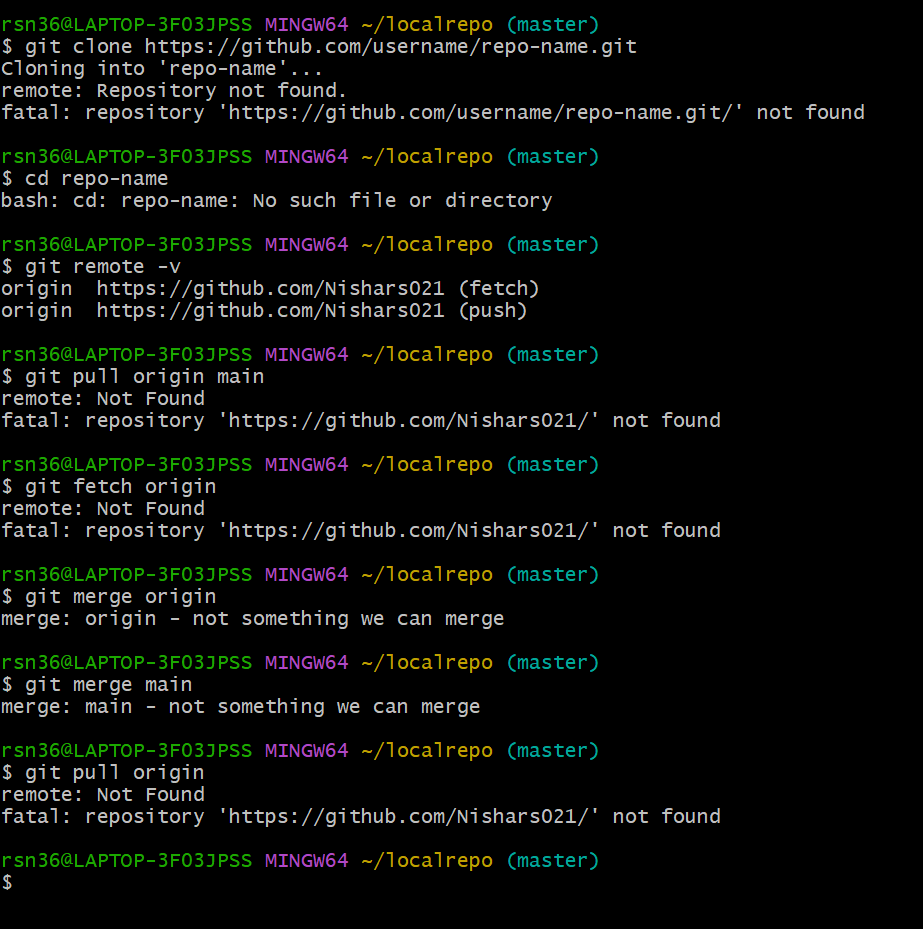
Case 1: Pushing a Local Repository to a Remote Repository

Scenario: You have a local project that you want to share by pushing it to a remote repository.



Case 2: Cloning and Pulling from a Remote Repository

Scenario: A remote repository exists that you want to work with locally. You'll clone it, observe changes made to the remote repository, and pull those changes to your local copy.



Handling Authentication & Potential Issues

Learning Outcomes  
1. Understanding how to connect local repositories to remote repositories  
2. Learning to push local changes to remote repositories  
3. Learning to pull remote changes to local repositories  
4. Managing typical remote repository workflows

Conclusion  
Working with remote repositories is essential for collaborative software development. It allows teams to share code, track changes, and maintain project history across multiple developers and workstations.

**Lab Exercise 9**: Collaborating using Pull Requests

Objective  
To understand and practice the GitHub pull request workflow, including creating branches, making changes, submitting pull requests, reviewing code, and merging changes.

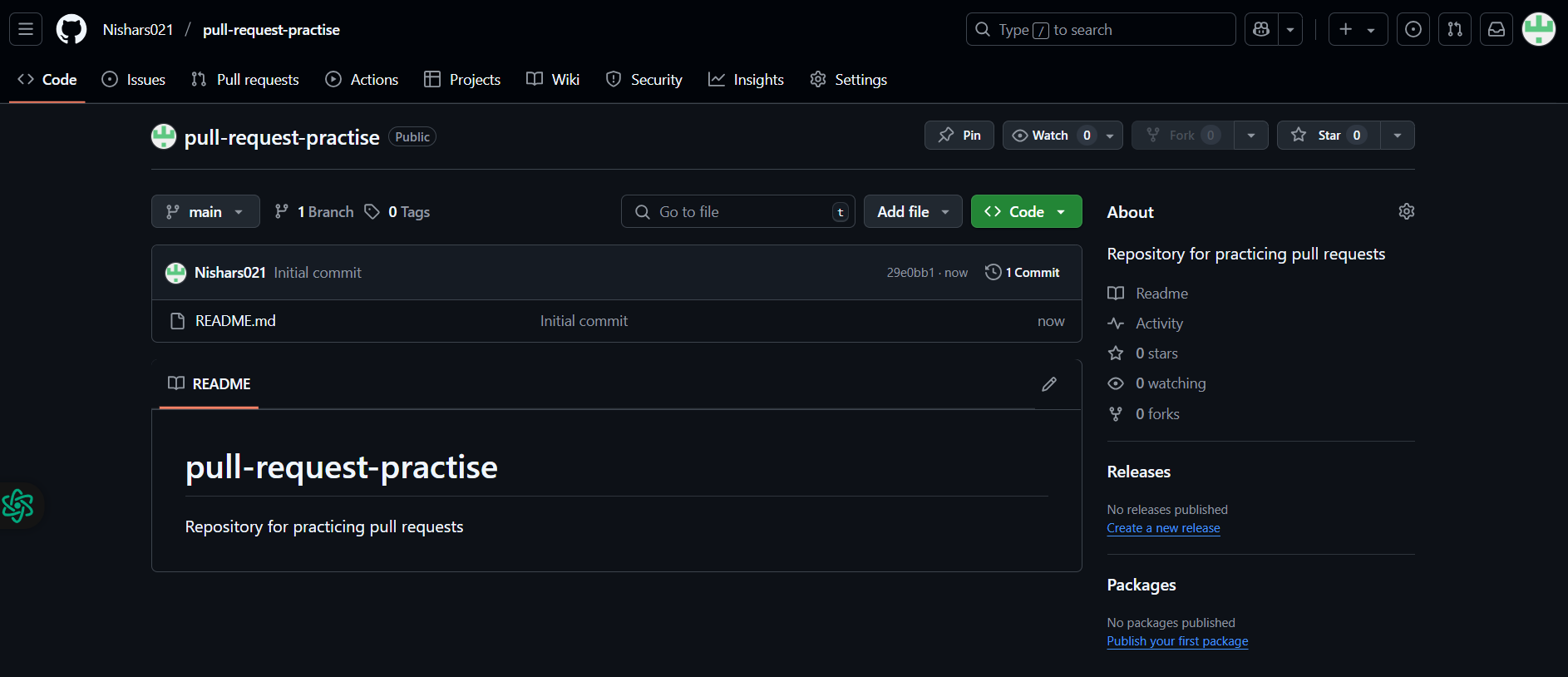
Prerequisites  
- GitHub account  
- Basic understanding of Git concepts  
- Web browser

Environment   
This lab was completed entirely through the GitHub web interface, no local Git installation was required.

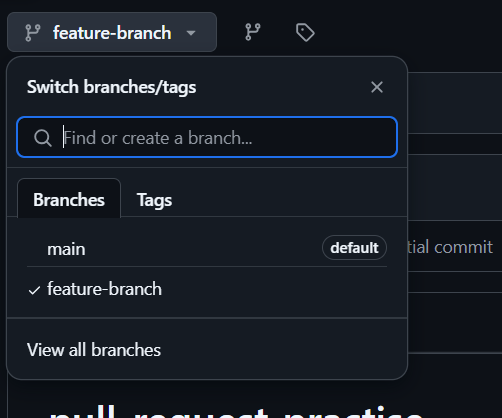
Case 1: Created a New Repository and Branch

Scenario: Set up a new project repository and create a feature branch for development.  
# No local commands needed - used GitHub web interface

Steps Completed and Output  
1. Created a new repository:  
   - Navigated to GitHub.com and signed in  
   - Clicked the "+" icon in the top-right corner  
   - Selected "New repository"  
   - Named it: "pull-request-practice"  
   - Description: "Repository for practicing pull requests"  
   - Visibility: Public  
   - Checked "Add a README file"  
   - Clicked "Create repository"  
   - Output: Screenshots of the above steps



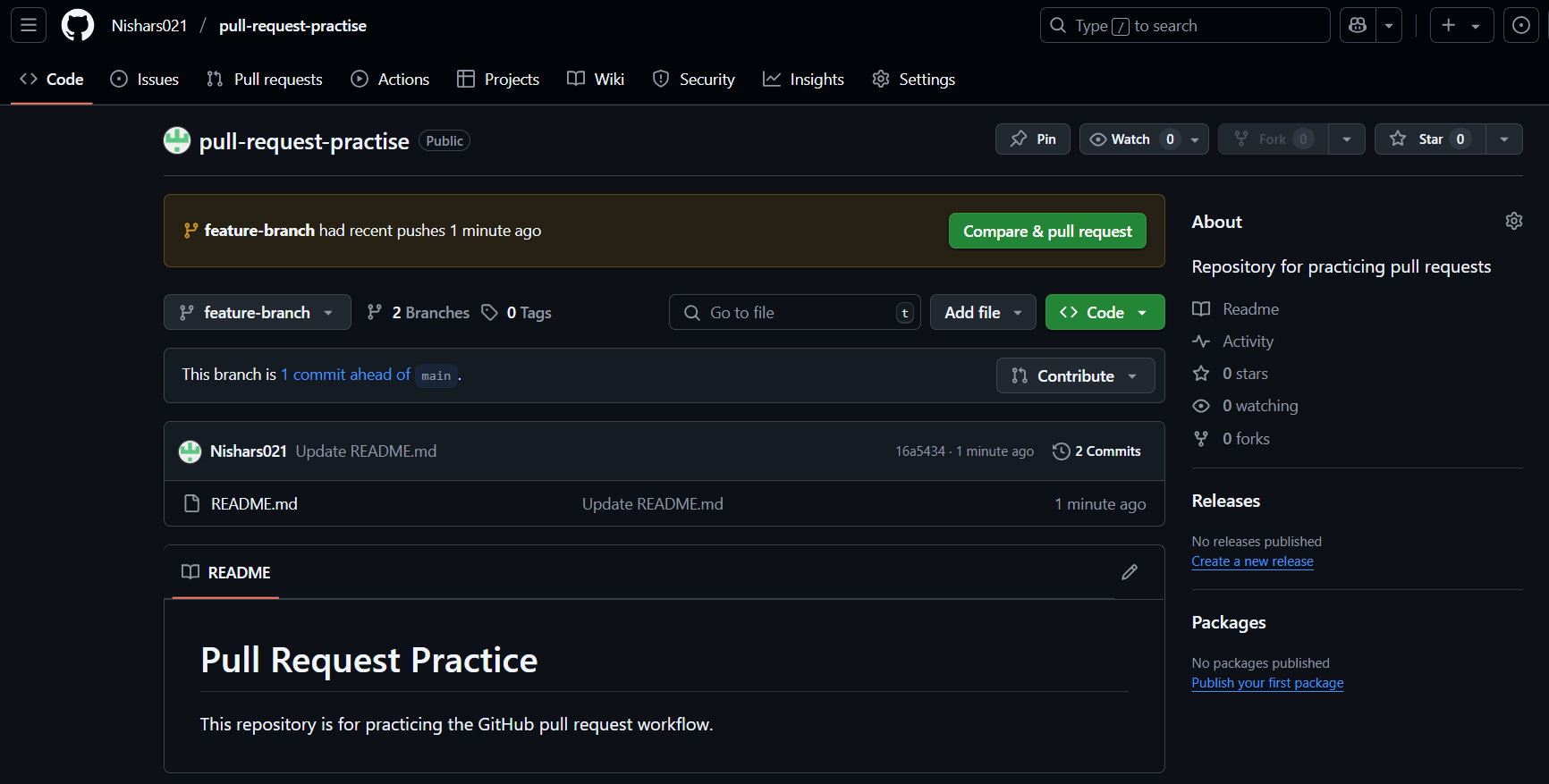
2. Created a new branch:  
   - In the repository, clicked the "main" branch dropdown  
   - Typed "feature-branch" in the search box  
   - Clicked "Create branch: feature-branch from 'main'"  
   - Output: Screenshots of the above steps



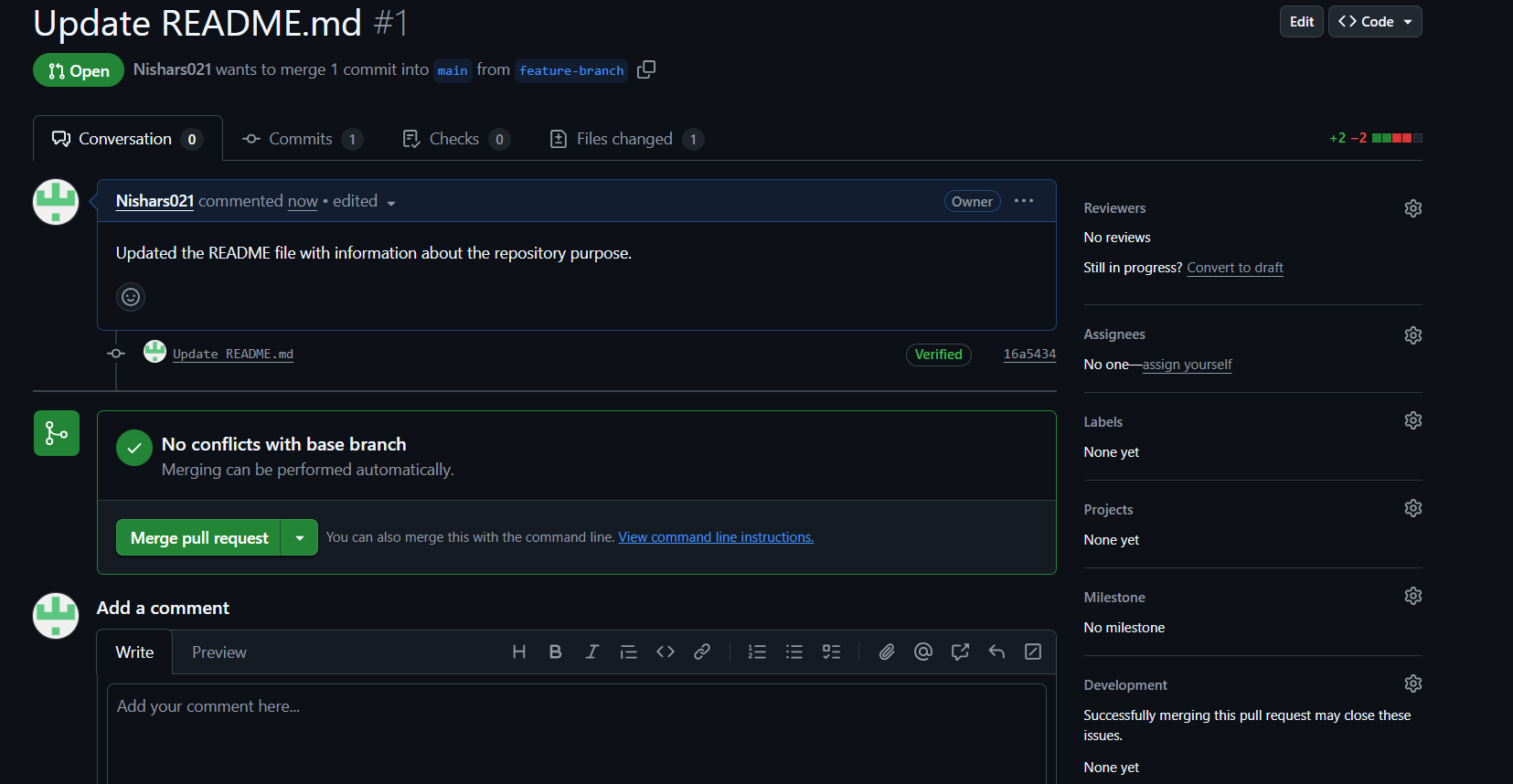
Case 2: Made Changes and Created a Pull Request

Scenario: Made code changes in my feature branch and created a pull request to propose merging these changes to the main branch.

Steps Completed and Output  
1. Edited a file:  
   - Navigated to the README.md file  
   - Clicked the pencil icon to edit  
   - Added content: "# Pull Request Practice\n\nThis repository is for practicing the GitHub pull request workflow."  
   - Scrolled down and added commit message: "Update README with project description"  
   - Selected "Commit directly to the feature-branch branch"  
   - Clicked "Commit changes"  
   - Output: Screenshots of the above steps



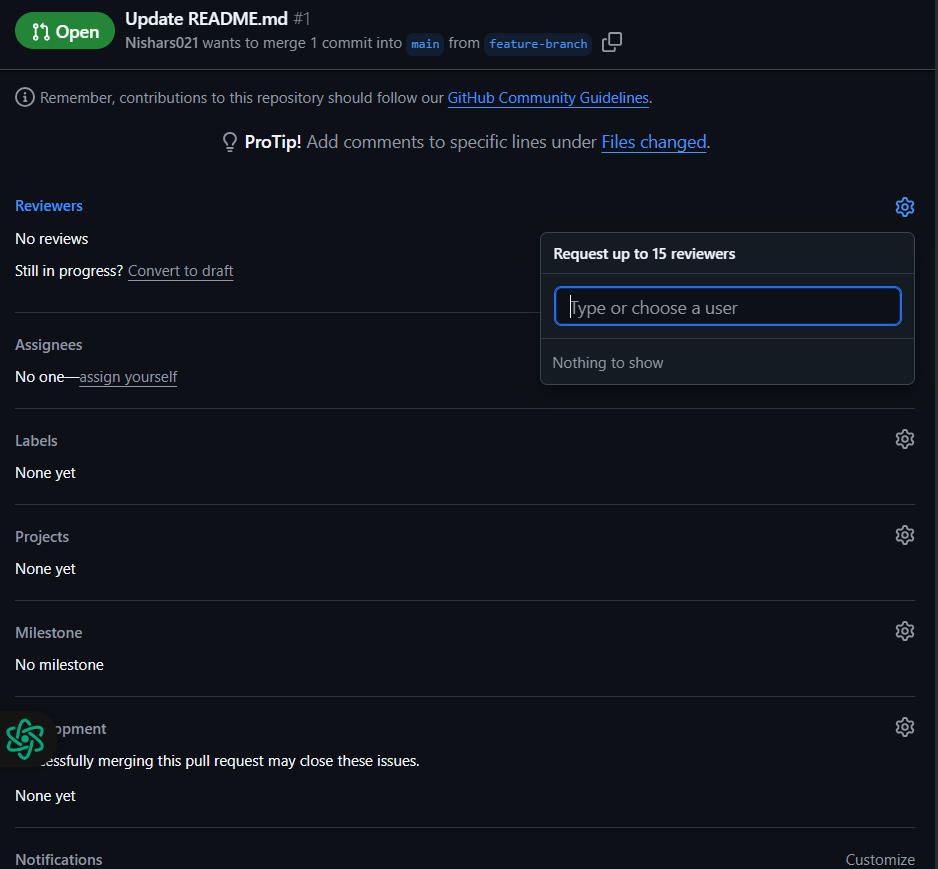
2. Created a pull request:  
   - Clicked on "Pull requests" tab  
   - Clicked "New pull request"  
   - Set "base: main" and "compare: feature-branch"  
   - Reviewed the changes  
   - Clicked "Create pull request"  
   - Title: "Add project description to README"  
   - Description: "Updated the README file with information about the repository purpose."  
   - Clicked "Create pull request"  
   - Output: Screenshots of the above steps



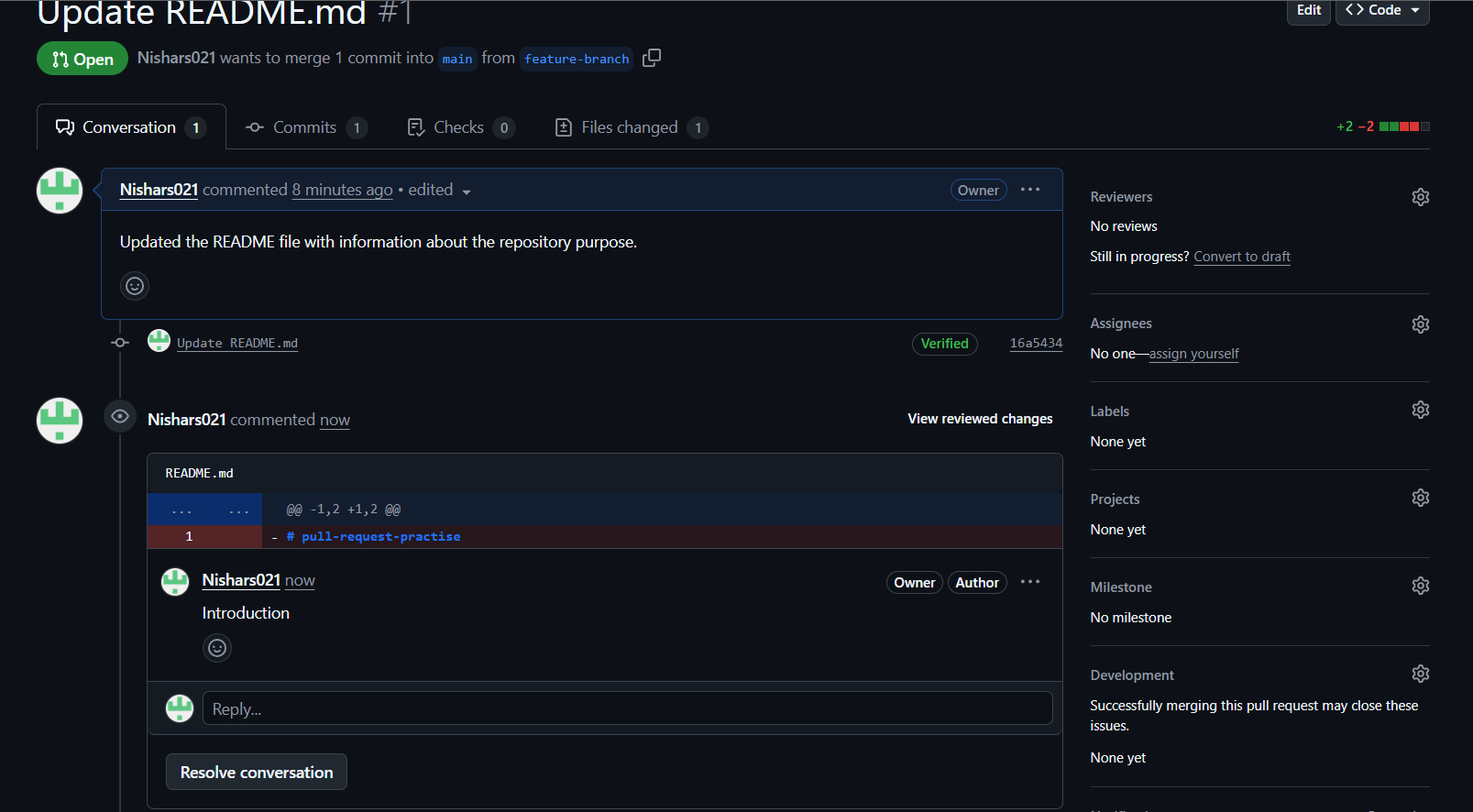
Case 3: Completed the Code Review Process

Scenario: My pull request needed to be reviewed before merging.

Steps Completed and Output  
1. Requested reviewers:  
   - In the pull request, clicked "Reviewers" on the right sidebar  
   - Searched for and selected teammates (since this was a solo project, I noted that in a team setting I would add team members here)  
   - Output: Screenshots of the above steps

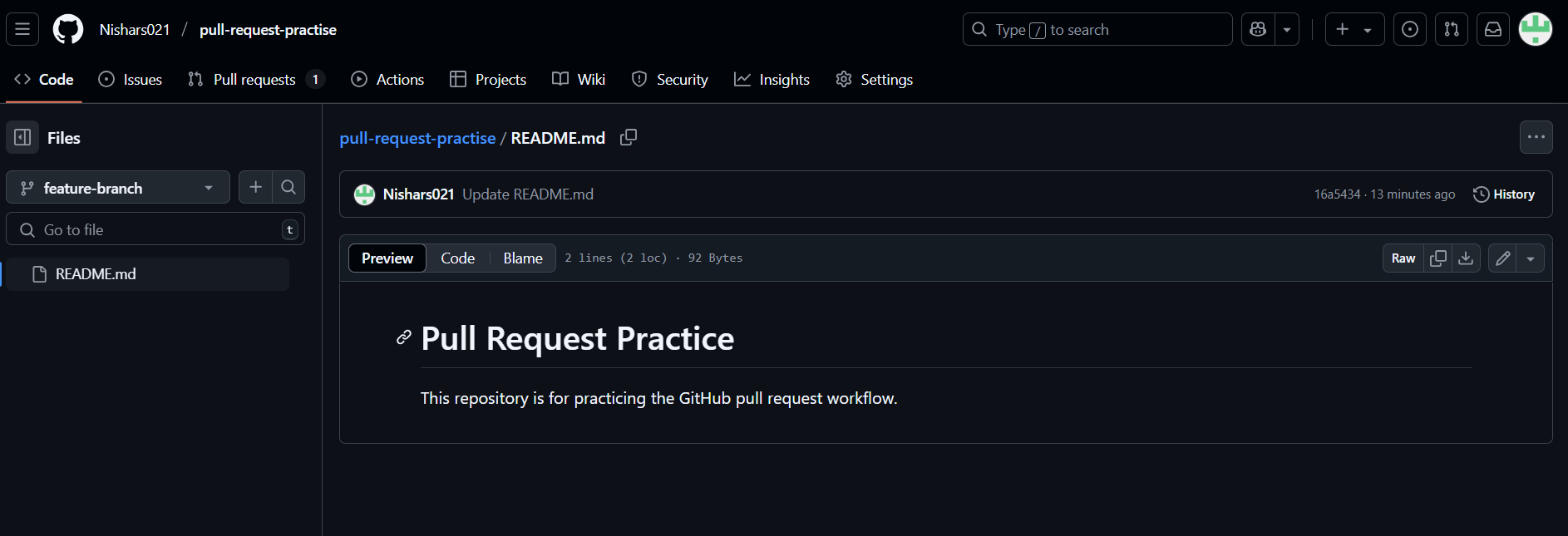


2. Added a review comment (simulated the reviewer role):  
   - In the "Files changed" tab, hovered over a line  
   - Clicked the "+" that appeared  
   - Added comment: "Consider adding more details about what specific PR workflows we'll practice"  
   - Clicked "Start a review"  
   - Clicked "Finish review"  
   - Selected "Request changes"  
   - Clicked "Submit review"  
   - Output: Screenshots of the above steps



3. Addressed review comments:  
   - Went back to the README.md file and clicked the pencil icon  
   - Added: "We will practice creating branches, making changes, creating pull requests, reviewing code, and merging changes."  
   - Commit message: "Add more details as requested in the review"  
   - Clicked "Commit changes"  
   - Output: Screenshots of the above steps

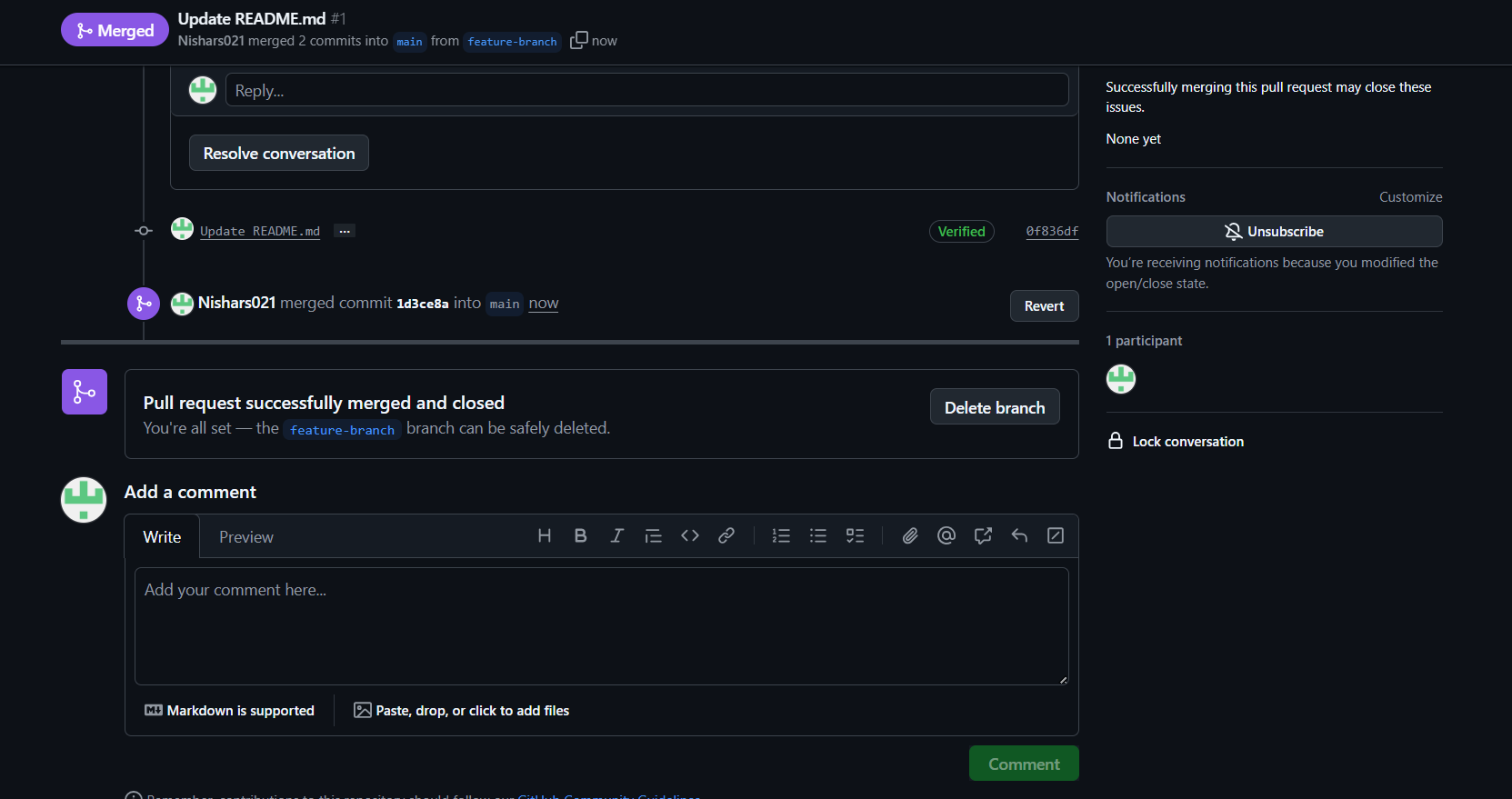
4. Approved the changes (simulated the reviewer role again):  
   - Went to "Files changed" tab  
   - Clicked "Review changes"  
   - Selected "Approve"  
   - Added comment: "Changes look good now, ready to merge."  
   - Clicked "Submit review"  
   - Output: Screenshots of the above steps



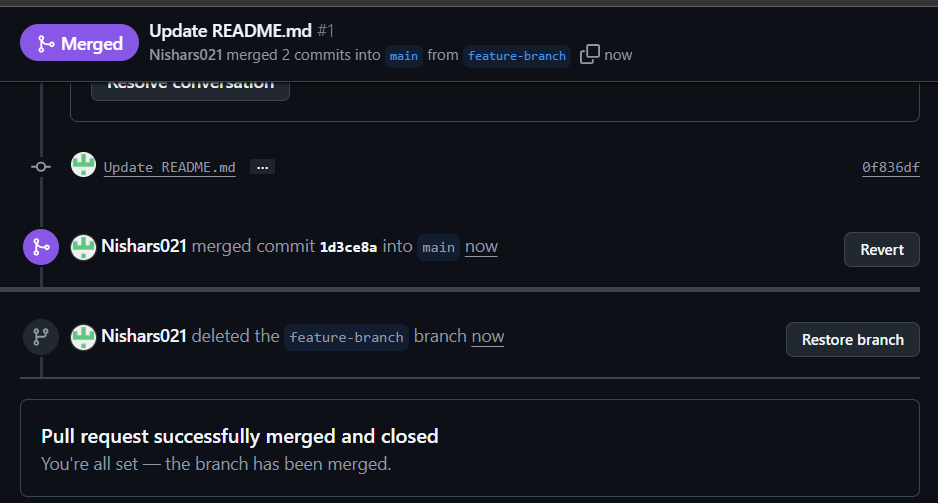
Case 4: Merged Changes

Scenario: The pull request was approved and ready to merge.

Steps Completed and Output  
1. Merged the pull request:  
   - Went to the "Conversation" tab  
   - Clicked "Merge pull request"  
   - Confirmed with "Confirm merge"  
   - Output: Screenshots of the above steps



2. Deleted the branch:  
   - Clicked "Delete branch" after merge  
   - Output: Screenshots of the above steps



Handled Potential Issues

Common Issue 1: Merge Conflicts  
- Problem: No merge conflicts occurred during my exercise  
- Knowledge Gained: I learned how to handle merge conflicts if they had occurred:  
  1. Click "Resolve conflicts" button that appears  
  2. Edit the file to resolve conflicts (remove conflict markers and keep desired changes)  
  3. Click "Mark as resolved"  
  4. Click "Commit merge"

Reviewers vs. Assignees - Understanding Gained

- Reviewers: People responsible for examining code changes and providing feedback  
  - I learned that reviewers should be added when technical feedback on implementation is needed  
  - Reviewers can approve or request changes

- Assignees: People responsible for the overall progress of the pull request  
  - Usually the person who created the PR or who is currently working on it  
  - Helps track ownership and responsibility for the pull request

Learning Outcomes Achieved  
1. Understood how to create and manage branches on GitHub  
2. Learned to create pull requests to propose changes  
3. Understood the code review process  
4. Learned how to address feedback and merge changes  
5. Understood the difference between reviewers and assignees

Conclusion  
I successfully completed the pull request workflow exercise. This process is fundamental to collaborative software development. It provides a structured way to propose, review, discuss, and merge changes, which improves code quality and team coordination. I am now comfortable with the GitHub pull request workflow and ready to apply these skills in team development environments.