**AMITY UNIVERSITY**



**SOUCE CODE LAB MANAGEMENT**

**LAB MANUAL**

**Submitted by: Rashmi Patil**

**Submitted to: Dr. Monit Kapoor**

**Submitted on:03.06.2025**

**ACKNOWLEDGMENT**

We would like to express our deepest gratitude to **Dr. Rajat** for his expert guidance, continuous support, and insightful feedback during the preparation of this **Source Code Management Lab Manual**. His vast knowledge, patience, and encouragement played a pivotal role in helping us understand and implement the concepts effectively.

We are also immensely thankful to **Amity University,Bangalore** for providing us with the necessary resources, infrastructure, and a conducive learning environment to carry out this lab work. The university’s emphasis on practical learning and industry-relevant skills greatly contributed to the successful completion of this manual.

Additionally, we extend our appreciation to our peers and colleagues for their valuable inputs and cooperation throughout this project.

This manual would not have been possible without the collective support of all those mentioned above.

**Thank you.**

INDEX

|  |  |  |
| --- | --- | --- |
| Sl.no | Content | Page no. |
| **1.** | **Basics of Linux** | **4-7** |
| **2.** | **Introduction- GitBash** | **8-11** |
| **3.** | **GitBash and Github** | **12-14** |
| **4.** | **File Creation with commit and push command** | **15-18** |
| **5.** | **Branches Creation** | **19-21** |
| **6.** | **Merging** | **22-25** |
| **7.** | **Open and Close Pull Request** | **26-28** |
| **8.** | **Complete Git Process** | **29-35** |
| **9.** | **Conclusion** | **36** |
| **10.** | **Bibliography** | **37** |

**1.BASICS OF LINUX**

Linux is a powerful, open-source operating system widely used in servers, desktops, and embedded systems. Below are the fundamental concepts you need to understand:

**1. Linux Architecture**

Linux follows a layered architecture:

* **Hardware** (CPU, RAM, Storage, etc.)
* **Kernel** (Core of Linux, manages hardware & processes)
* **Shell** (Command-line interface like Bash, Zsh)
* **Applications** (GUI & CLI tools like Firefox, LibreOffice)

**2. Linux File System Hierarchy**

* Linux organizes files in a tree-like structure starting from the root (/). Key directories:

| **Directory** | **Purpose** |
| --- | --- |
| / | Root directory |
| /bin | Essential user binaries (commands like ls, cp) |
| /etc | System configuration files |
| /home | User personal files |
| /var | Variable data (logs, databases) |
| /usr | User programs & libraries |
| /tmp | Temporary files |
| /dev | Device files (hardware representation) |

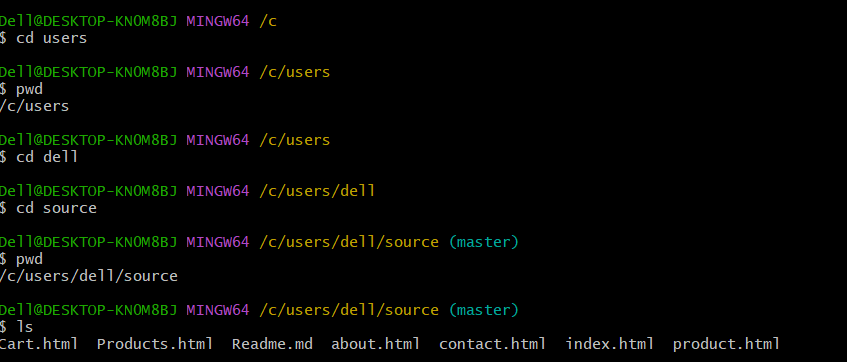
**3. Basic Linux Commands**

**File & Directory Operations**

| **Command** | **Description** |
| --- | --- |
| **ls** | **List files/directories** |
| **cd** | **Change directory** |
| **pwd** | **Print current directory** |
| **mkdir** | **Create a directory** |
| **rm** | **Remove files/directories** |
| **cp** | **Copy files** |
| **mv** | **Move/rename files** |
| **cat** | **Display file content** |
| **nano / vim** | **Text editors** |

**System Information**

| **Command** | **Description** |
| --- | --- |
| **uname -a** | **Show system info** |
| **df -h** | **Disk space usage** |
| **free -h** | **Memory usage** |
| **top / htop** | **Process monitoring** |

****

**4. Users & Groups**

* Linux is multi-user, meaning multiple users can work simultaneously.
* Root user (sudo) has admin privileges.
* **Commands:**
  + useradd (Create user)
  + passwd (Set password)
  + usermod (Modify user)
  + groups (List user groups)

**5. Processes & Services**

* A **process** is a running program.
* Commands:
  + ps (List processes)
  + kill (Terminate a process)
  + systemctl (Manage services, e.g., systemctl start nginx)

**6. Package Management**

Different Linux distros use different package managersdistros use different package managers:

| **Distro** | **Package Manager** | **Install Example** |
| --- | --- | --- |
| Ubuntu/Debian | apt | sudo apt install firefox |
| Fedora/RHEL | dnf / yum | sudo dnf install firefox |
| Arch Linux | pacman | sudo pacman -S firefox |

**7. Networking Basics**

| **Command** | **Description** |
| --- | --- |
| ping | Check connectivity |
| ifconfig / ip a | Network interface info |
| ssh | Remote login |
| scp | Secure file transfer |

**8. Shell Scripting Basics**

Linux allows automation using shell scripts (.sh files).  
Example (hello.sh)

**2.INTRODUCTION TO GITBASH**

Git Bash is a command-line interface (CLI) for Windows that provides a Unix-like shell environment, allowing you to run **Git commands** and **Linux-style terminal commands** on Windows. It comes bundled with **Git for Windows** and uses **Bash (Bourne Again Shell)**.

**1. Installing Git Bash**

1. Download **Git for Windows** from [git-scm.com](https://git-scm.com/downloads).
2. Run the installer (keep default settings).
3. After installation, open **Git Bash** from the Start Menu.

**2. Basic Git Bash Commands (With Examples)**

Git Bash supports both **Git commands** and **Linux commands**. Below are some essential commands:

**A. Navigation (Linux Commands)**

| **Command** | **Example** | **Description** |
| --- | --- | --- |
| pwd | pwd | Print current directory |
| ls | ls -l | List files (with details) |
| cd | cd Documents | Change directory |
| mkdir | mkdir project | Create a folder |
| touch | touch file.txt | Create a file |
| cat | cat file.txt | Display file contents |

**Example:**

bash

pwd# Shows current directory (e.g., /c/Users/YourName)

ls-la# Lists all files (including hidden)

cd Desktop # Moves to Desktop

mkdir my\_project # Creates a folder

touch README.md # Creates a file

cat README.md # Displays file content

**B. Git Commands**

| **Command** | **Example** | **Description** |
| --- | --- | --- |
| git init | git init | Initialize a Git repo |
| git clone | git clone https://github.com/user/repo.git | Clone a repository |
| git add | git add file.txt | Stage changes |
| git commit | git commit -m "message" | Commit changes |
| git push | git push origin main | Push to remote repo |
| git pull | git pull origin main | Pull latest changes |

**Example:**

bash

git init # Creates a new Git repo

gitadd.# Stages all changes

git commit -m"First commit"# Commits changes

git remote add origin <repo-url># Links to a remote repo

git push -u origin main # Pushes to GitHub

**3. Customizing Git Bash**

**A. Change Default Directory**

By default, Git Bash opens in C:\Users\YourName. To change this:

1. Right-click Git Bash shortcut → **Properties**.
2. In **Start in**, enter your preferred path (e.g., C:\Projects).

**B. Aliases (Shortcuts)**

Add shortcuts in ~/.bashrc:

bash

echo"alias gs='git status'">> ~/.bashrc

source ~/.bashrc # Reloads config

Now, gs will run git status.

**4. Git Bash vs Windows CMD/PowerShell**

| **Feature** | **Git Bash** | **CMD** | **PowerShell** |
| --- | --- | --- | --- |
| **Shell Type** | Bash | Command Prompt | PowerShell |
| **Linux Commands** | ✅ Yes | ❌ No | ⚠️ Some |
| **Git Integration** | ✅ Built-in | ❌ No | ❌ No |
| **Scripting** | Bash scripts | Batch files | PowerShell scripts |

**5. Example Workflow**

**Clone a Repo, Make Changes, and Push**

Bash

git clone https://github.com/user/repo.git # Clone repo

cd repo # Enter repo

nano file.txt # Edit file (Ctrl+X to save)

git add file.txt # Stage changes

git commit -m "Updated file" # Commit

git push origin main # Push to GitHub

**6. Useful Tips**

* Press Tab for **auto-completion**.
* Use Ctrl + C to **cancel** a running command.
* history shows your command history.clear
* cleans the terminal.

**3.GITBASH AND GITHUB**

**1. Git Bash on Windows**

Git Bash is a **command-line interface** that brings Git and Unix-style commands to Windows.

**What it does:**

* Lets you run **Git commands** (like git commit, git push)
* Provides **Linux-style terminal commands** (like ls, cd, grep)
* Works locally on your machine (no internet needed for basic Git operations)

**Key Features:**

✔️ Native Git support  
✔️ Bash shell environment  
✔️ Works offline for local repositories  
✔️ Includes common Unix tools (ssh, curl, vim)

**Installation:**

1. Download from [git-scm.com](https://git-scm.com/downloads)
2. Run installer (keep default settings)
3. Launch from Start Menu

**Basic Usage Examples:**

Bash

# Navigate to folder

cd /c/projects

# Initialize Git repository

git init

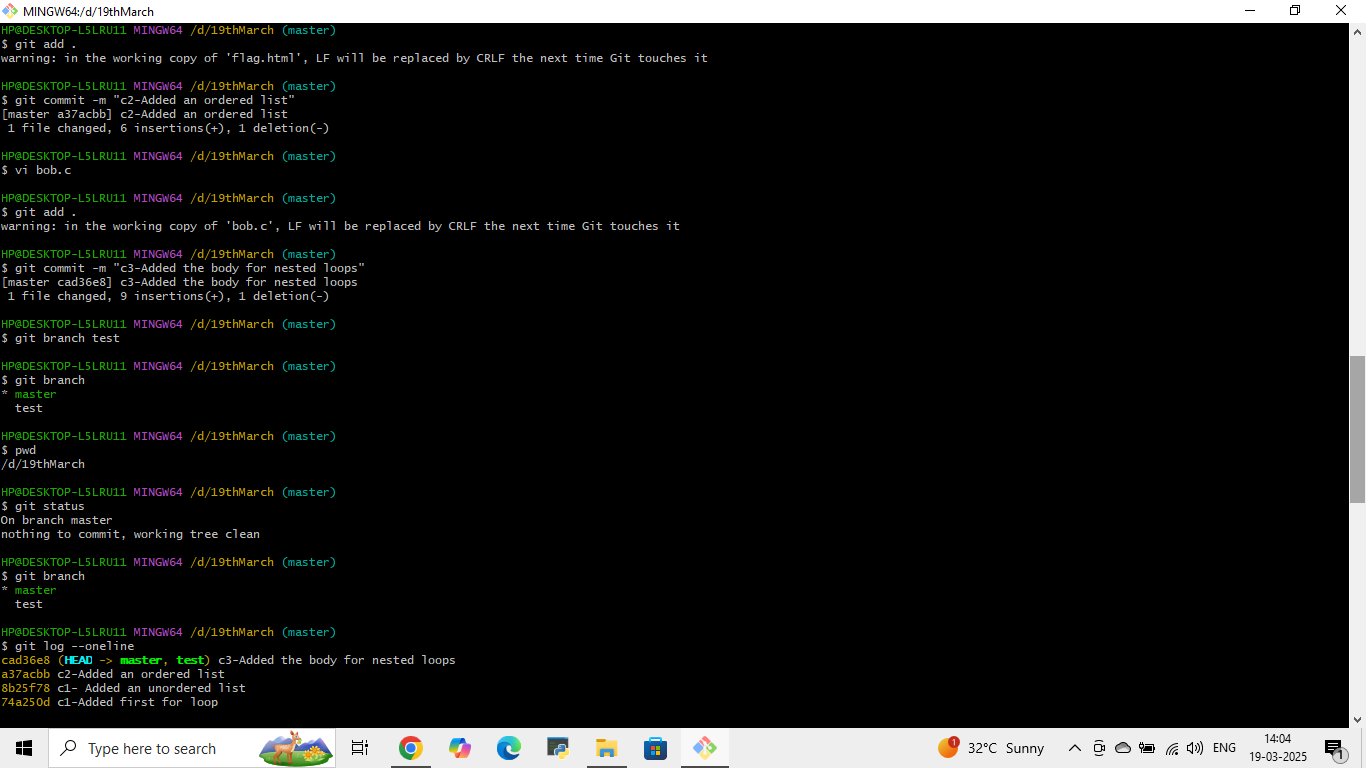
# Check status

git status

# Create and edit files

touch index.html

nano index.html # Edit file



**2. GitHub on Windows**

GitHub is a **cloud-based platform** for hosting Git repositories.

**What it does:**

* Stores your code in the cloud
* Enables collaboration with others
* Provides issue tracking and project management

**Key Features:**

🌐 Remote repository hosting  
👥 Team collaboration tools  
🔀 Pull request workflow  
🚀 GitHub Desktop (GUI alternative)

**How to Use GitHub with Git Bash:**

1. **Create a GitHub account** at [github.com](https://github.com/)
2. **Create a new repository** on GitHub
3. **Connect your local repo** (in Git Bash):

Bash

# Clone an existing repository

git clone <https://github.com/username/repo.git>

# Or connect local repo to GitHub

git remote add origin <https://github.com/username/repo.git>

git push -u origin main

**Typical Workflow:**

Bash

# Make changes locally

git add .

git commit -m "Update feature"

# Push to GitHub

git push origin main

# Pull latest changes

git pull origin main

**3. Key Differences**

| **Feature** | **Git Bash** | **GitHub** |
| --- | --- | --- |
| **Location** | Local machine | Cloud |
| **Internet Needed?** | No (for local ops) | Yes |
| **Primary Use** | Version control commands | Code hosting/collaboration |
| **Interface** | Command line | Web interface/GUI |

**4. Pro Tips for Windows Users**

1. **Use GitHub Desktop** for a graphical interface
2. **Generate SSH keys** for secure connections:

Bash

keygen -t ed25519 -C "your\_email@example.com"

**Set your identity** in Git Bash:

Bash

git config --global user.name "Your Name"

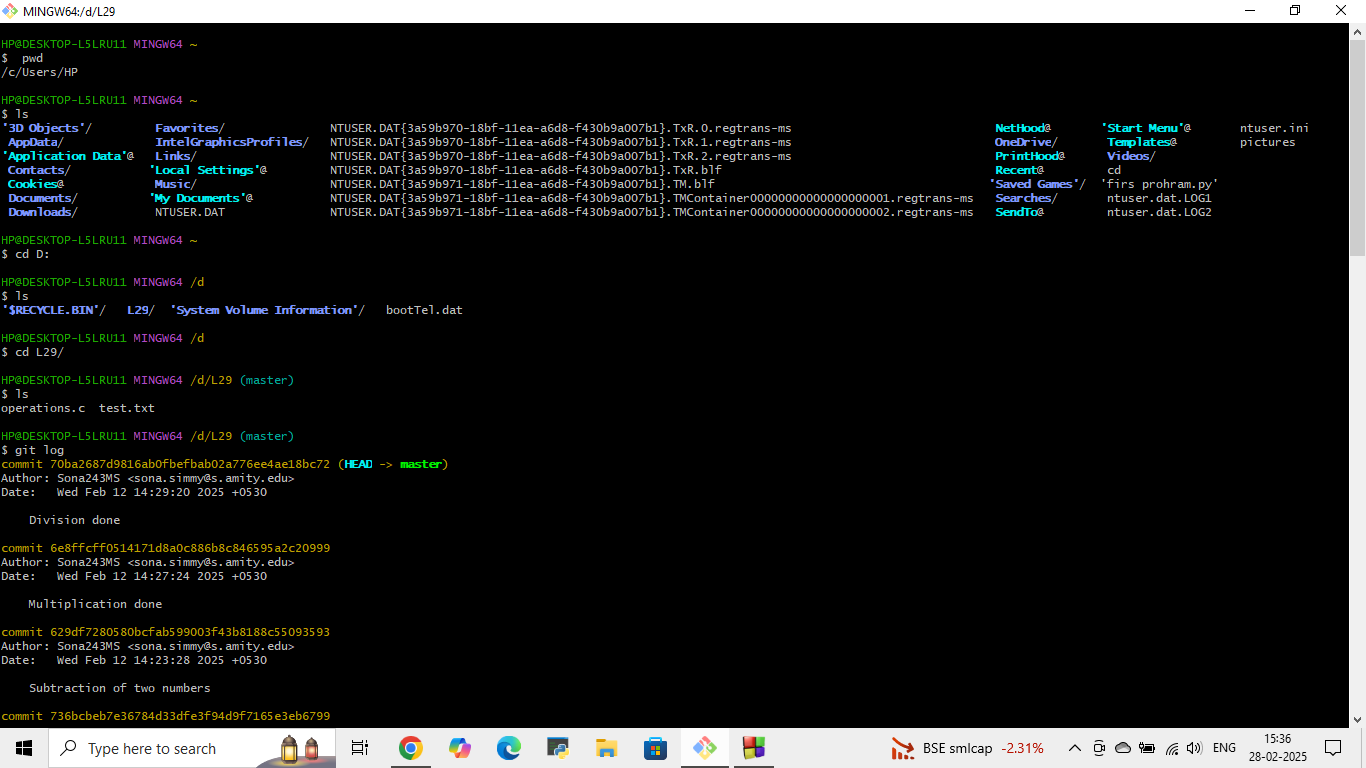
git config --global user.email "your@email.com"

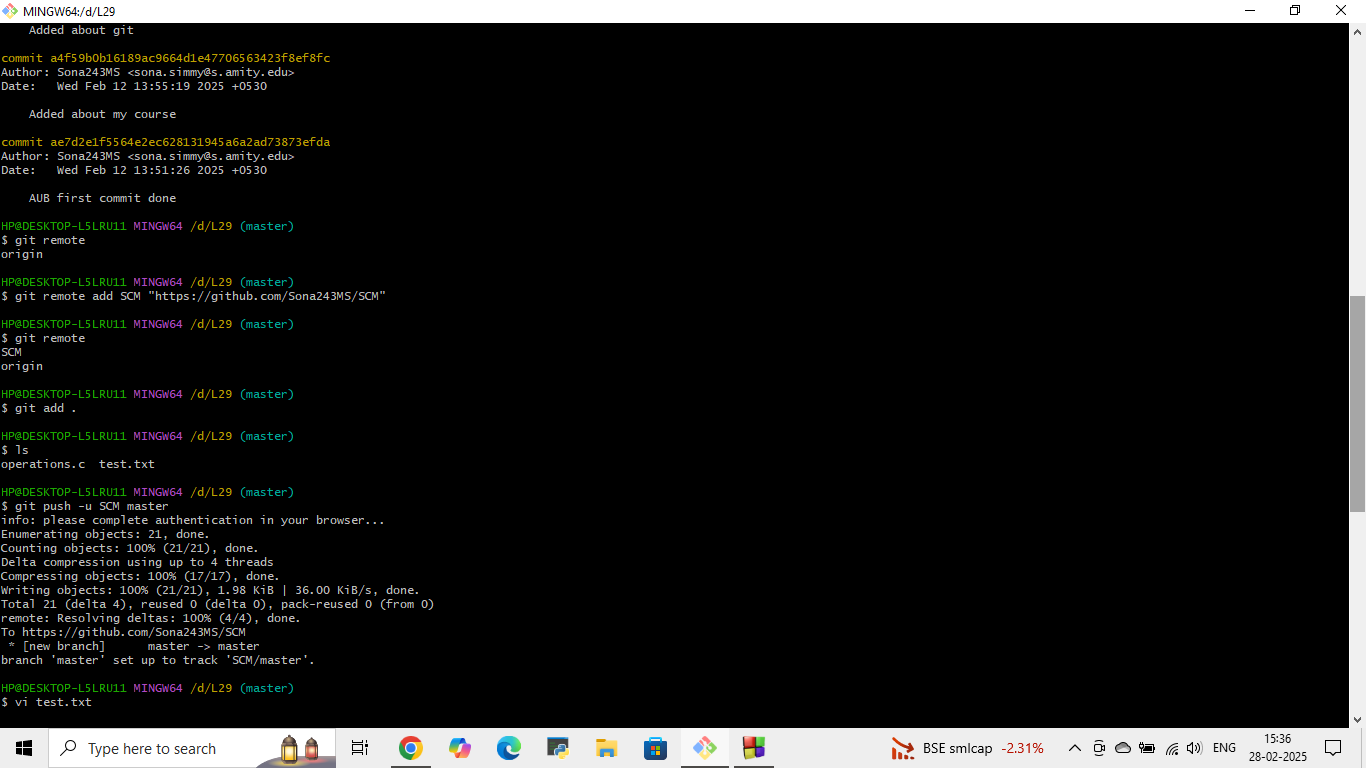
**5. When to Use Each**

* Use **Git Bash** when:
  + Working with Git locally
  + Running scripts/automation
  + Need Linux-style commands
* Use **GitHub** when:
  + Storing code in the cloud
  + Collaborating with a team
  + Managing project issues

4.File Creation with Commit and Push Command

Here's a step-by-step guide to creating files, committing changes, and pushing to GitHub using Git Bash on Windows:





**1. Set Up Git (One-Time Configuration)**

Before starting, configure your Git identity:

Bash

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

**2. Create a New File and Push to GitHub**

**Step 1: Initialize a New Git Repository**

Bash

# Navigate to your project folder

cd /c/Users/YourName/projects

# Create a new directory (optional)

mkdir my-project

cd my-project

# Initialize Git

git init

**Step 2: Create a New File**

Bash

# Create a file (multiple methods)

touch README.md # Creates empty file

echo "# My Project" > README.md # Creates file with content

notepad README.md # Opens in Notepad to edit

**Step 3: Check Status & Stage Changes**

Bash

git status # Shows untracked/modified files

git add README.md # Stages the file

git add . # Stages ALL new/changed files

**Step 4: Commit Changes**

Bash

git commit -m "Add README file" # Commits with a message

**Step 5: Connect to GitHub & Push**

Bash

# Create a new repository on GitHub (no README) and copy its URL

git remote add origin <https://github.com/your-username/repo-name.git>

git branch -M main # Renames default branch to 'main'

git push -u origin main # Pushes to GitHub

**3. Update an Existing File & Push Changes**

**Step 1: Modify the File**

bash

# Edit the file (choose one method)

nano README.md # Edit in Nano

notepad README.md # Edit in Notepad

echo"New line">> README.md # Append text via command line

**Step 2: Commit & Push Updates**

bash

git status # Verify changes

gitadd README.md # Stage the file

git commit -m"Update README"

git push origin main # Push to GitHub

**4. Common Issues & Fixes**

**Error: "Remote origin already exists"**

bash

git remote remove origin # Remove old origin

git remote add origin NEW-URL

**Error: "Updates were rejected"**

Bash

git pull origin main # Pull latest changes first

git push origin main # Try pushing again

**Forgot to Add Files Before Commit?**

bash

gitadd forgotten-file.txt

git commit --amend# Updates the last commit

git push -f origin main # Force push (use cautiously!)

**5. Summary Cheat Sheet**

| **Action** | **Command** |
| --- | --- |
| Create file | touch file.txt or echo "text" > file.txt |
| Stage file | git add file.txt or git add . |
| Commit | git commit -m "Message" |
| Push to GitHub | git push origin main |
| Check status | git status |

**Pro Tip:** Use git log to view your commit history!

**5.BRANCH CREATION**

**1. Creating a New Branch**

**View Existing Branches**

bash

git branch # Lists all local branches

git branch -a# Lists ALL branches (including remote)

**Create a New Branch**

Bash

git branch new-feature # Creates branch but doesn't switch

**Create & Switch to Branch (Recommended)**

Bash

git checkout -b new-feature # Creates and switches

# OR (newer Git versions)

git switch -c new-feature # Modern alternative

**2. Switching Between Branches**

bash

git checkout main # Switch to 'main' branch

git checkout new-feature # Switch back to feature branch

# OR

git switch main # Newer syntax

**3. Making Changes in a Branch**

1. Switch to your branch:

Bash

git checkout new-feature

1. Make changes (create/modify files):

bash

touch feature-file.txt

nano feature-file.txt

1. Commit changes:

bash

gitadd.

git commit -m"Added new feature"

**4. Pushing a Branch to GitHub**

bash

git push -u origin new-feature # First push (sets upstream)

# Subsequent pushes:

git push

**5. Common Branching Workflow**

bash

# Start new feature

git checkout -b new-feature

# Make changes and commit

gitadd.

git commit -m"Implemented feature"

# Push to GitHub

git push -u origin new-feature

# Create Pull Request on GitHub

# After approval:

git checkout main

git pull origin main # Get latest changes

git merge new-feature

git push origin main

# Clean up

git branch -d new-feature

git push origin --delete new-feature

**6. Pro Tips for Branch Management**

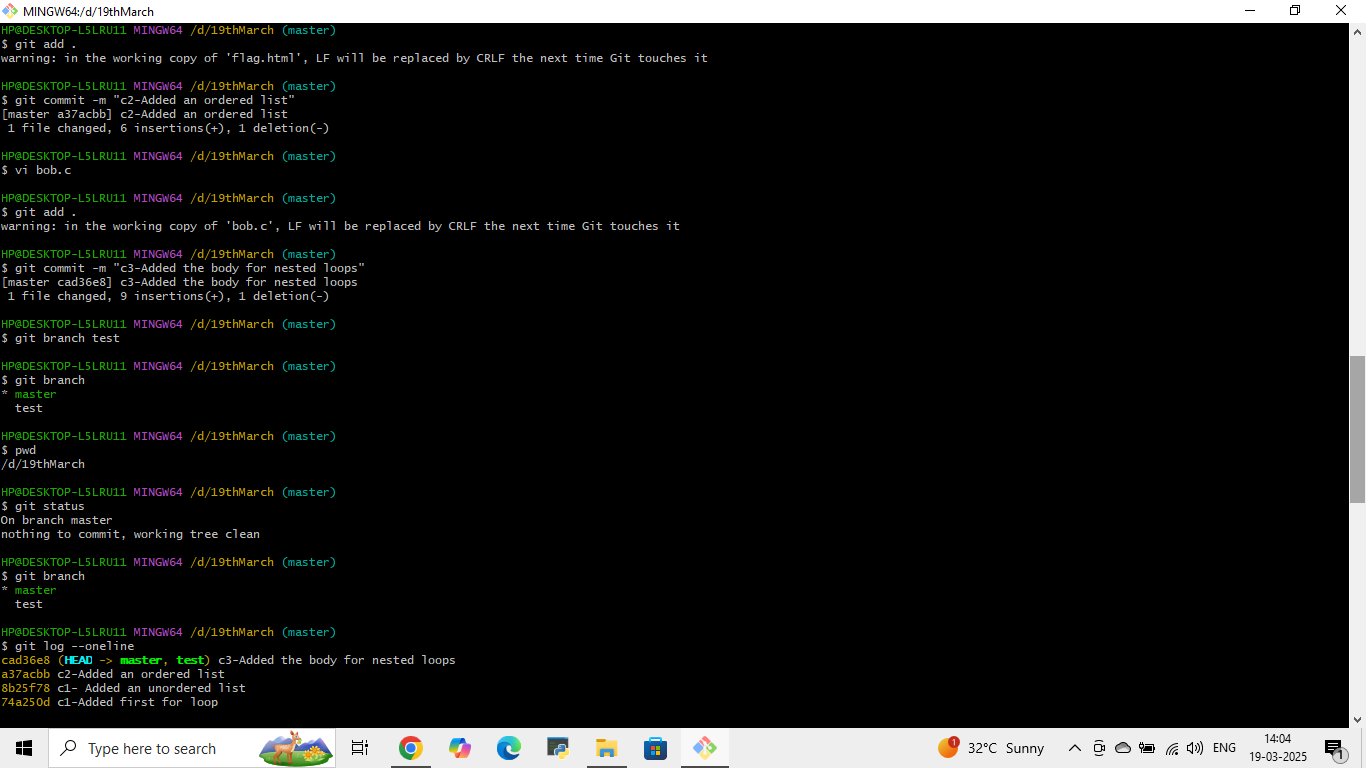
1. **Keep branches short-lived** - Merge quickly to avoid conflicts
2. **Pull latest changes** before creating new branches:

Bash

git checkout main

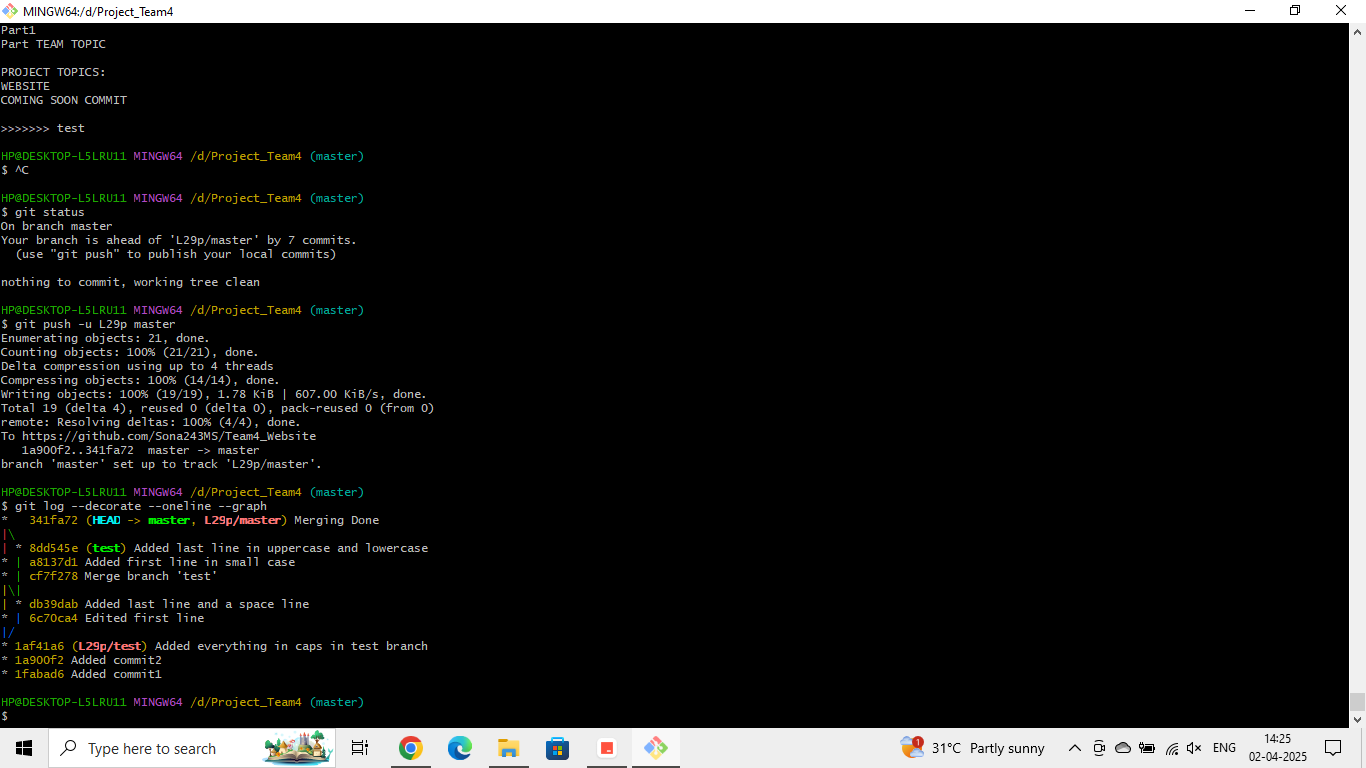
git pull origin main

1. **Name branches clearly** (e.g., fix/login-button instead of patch1)
2. **Use GitHub Desktop** for visual branch management if preferred



**6.MERGING**

Merging combines changes from different branches in Git. Here's how to do it properly in Windows using Git Bash:



**1. Basic Merging Workflow**

**Step 1: Ensure Your Main Branch is Updated**

bash

git checkout main # Switch to main branch

git pull origin main # Get latest changes

**Step 2: Merge Your Feature Branch**

bash

git merge feature-branch # Merge changes into main

**Step 3: Resolve Conflicts (If Any)**

1. Git will mark conflicted files
2. Open files in editor (VS Code, Notepad++, etc.)
3. Look for conflict markers:

<<<<<<< HEAD

Your current changes

=======

Incoming changes

>>>>>>> feature-branch

1. Edit to keep what you want, then save

**Step 4: Complete the Merge**

bash

gitadd.# Mark conflicts as resolved

git commit # Git will auto-generate merge message

git push origin main # Push merged changes

**2. Types of Merges**

**Fast-Forward Merge (Linear History)**

Occurs when no divergent changes exist:

Bash

git merge --ff-only feature-branch # Safe merge (fails if not possible)

**Three-Way Merge (Non-Linear History)**

Creates a new merge commit:

bash

git merge --no-ff feature-branch # Always creates merge commit

**Squash Merge (Clean History)**

Combines all feature commits into one:

bash

git merge --squash feature-branch

git commit -m"Combined feature changes"

**3. Handling Merge Conflicts**

**View Conflicts**

bash

git status # Shows conflicted files

**Use VS Code (Recommended for Windows)**

bash

code .# Opens VS Code with conflict markers

* Use VS Code's Git GUI to resolve conflicts visually

**Abort a Merge**

bash

git merge --abort# Reverts to pre-merge state

**4. Best Practices**

1. **Always pull latest changes** before merging:

Bash

git checkout main

git pull origin main

1. **Keep branches short-lived** to minimize conflicts
2. **Test before merging**:

Bash

git checkout feature-branch

git merge main # Test merge in your branch first

1. **Use merge tools** for complex conflicts:

bash

git mergetool # Launches configured diff tool

**5. Common Merge Commands Cheat Sheet**

| **Command** | **Purpose** |
| --- | --- |
| git merge branch-name | Basic merge |
| git merge --abort | Cancel ongoing merge |
| git diff --name-only --diff-filter=U | List conflicted files |
| git reset --hard | Discard all merge changes |
| git log --merge | Show commits causing conflicts |

**6. Visual Merge Tools for Windows**

1. **VS Code** (Built-in Git support)
2. **GitKraken** (GUI client)
3. **Meld** (Standalone diff tool)
4. **TortoiseGit** (Windows shell integration)

To configure VS Code as merge tool:

bash

git config --global merge.tool vscode

git config --global mergetool.vscode.cmd "code --wait $MERGED"**7.OPEN AND CLOSE PULL REQUEST**

A **Pull Request (PR)** in Git is a way to propose changes from your local branch to a remote repository (e.g., GitHub, GitLab, Bitbucket). Below is a step-by-step guide on how to create a PR using **Git Bash** (command line) and GitHub, with explanations and simulated "screenshots" (text-based representations).

**Step-by-Step: Creating a Pull Request via Git Bash**

**Prerequisites**

1. You have a **GitHub account** and a repository.
2. Git Bash is installed on your system.
3. You’ve cloned the repo and made changes locally.

**1. Create a New Branch**

Never commit directly to main/master. Instead, create a feature/bugfix branch:

* -b creates a new branch.
* feature/login-page is an example branch name.

**2. Make Changes & Commit**

* Make your code changes (e.g., edit login.html).
* Stage and commit:

**3. Push the Branch to GitHub**

Push your local branch to the remote repository:

**4. Create a Pull Request on GitHub**

After pushing, Git Bash will show a link to create a PR. Open it in a browser:

**Key Fields**:

* **Base**: Target branch (usually main).
* **Compare**: Your feature branch.
* **Description**: Explain changes (like in the PR template earlier).

**5. Review & Merge**

1. Team members review the PR on GitHub.
2. If approved, click **"Merge Pull Request"**.

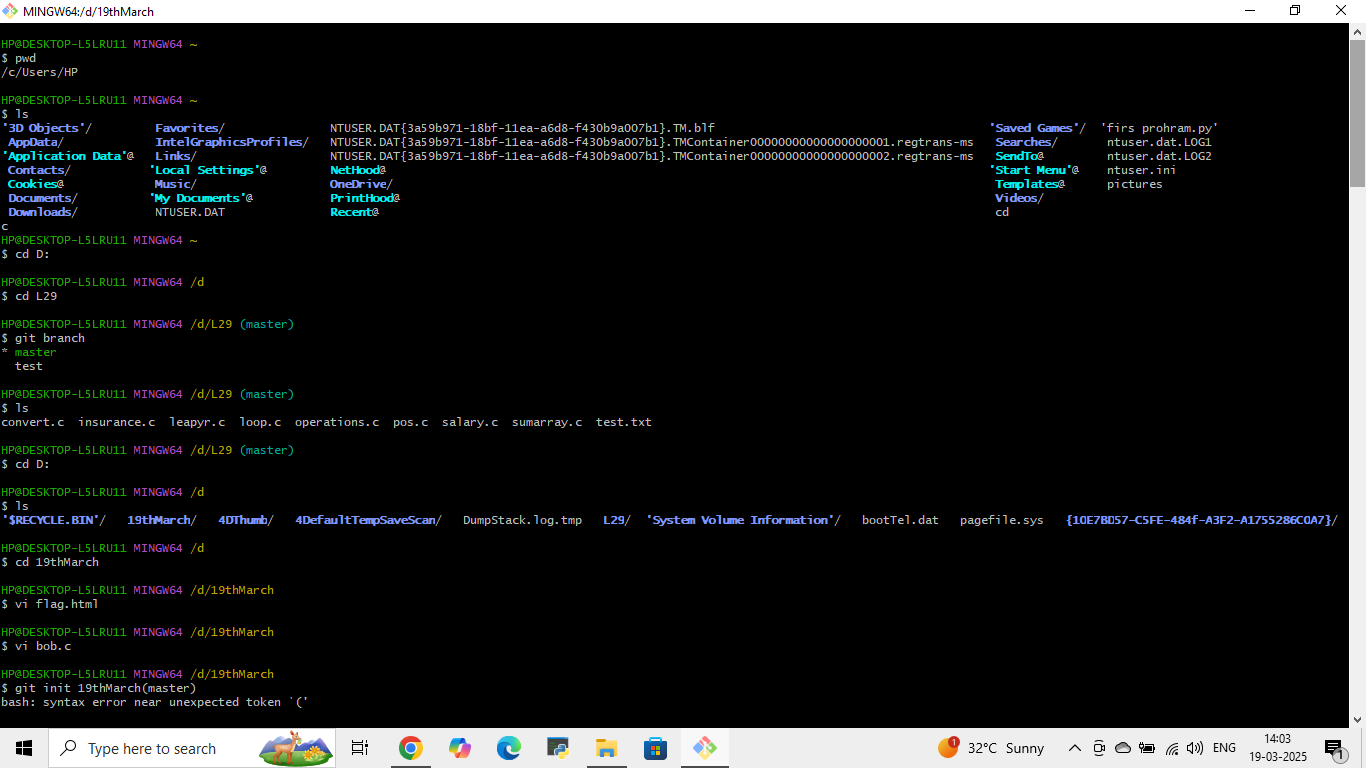
**6. Cleanup (Optional)**

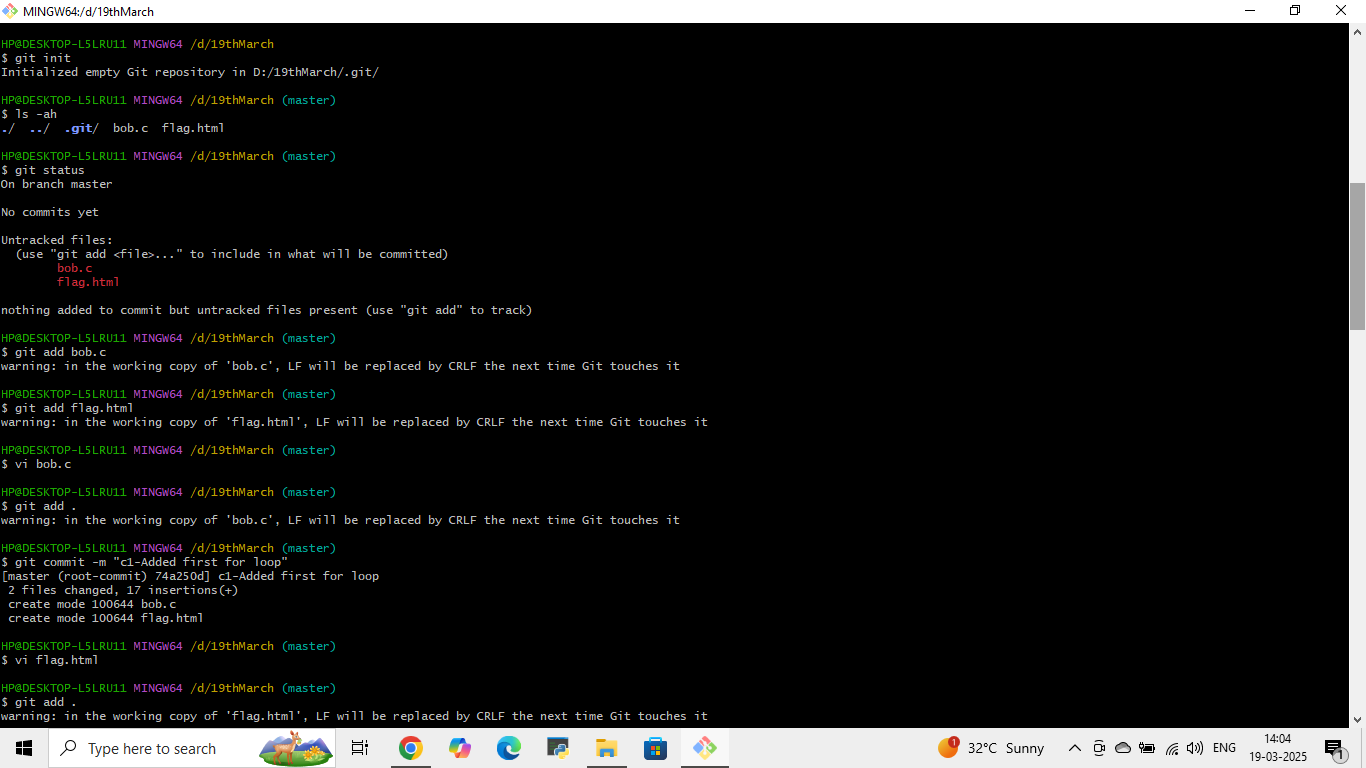
After merging, delete the remote and local branches:

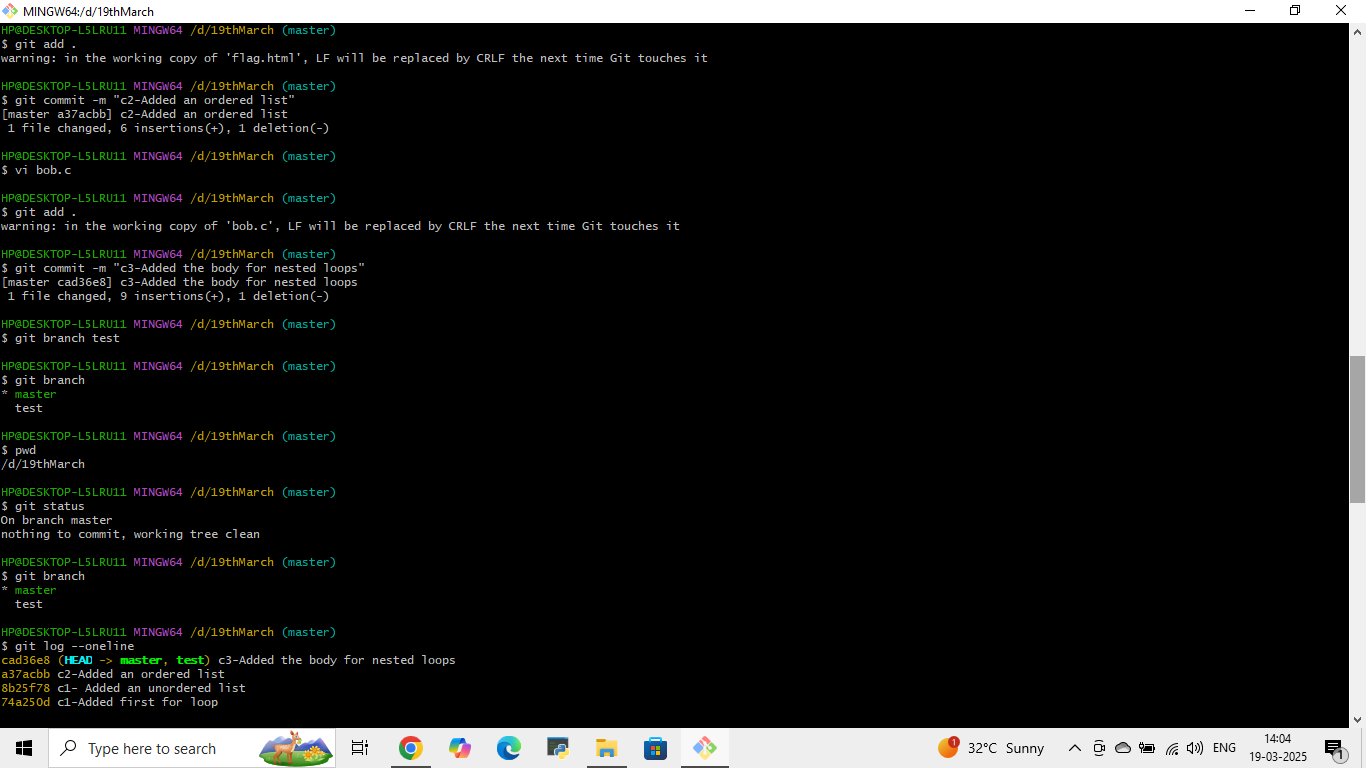
**Why Use Pull Requests?**

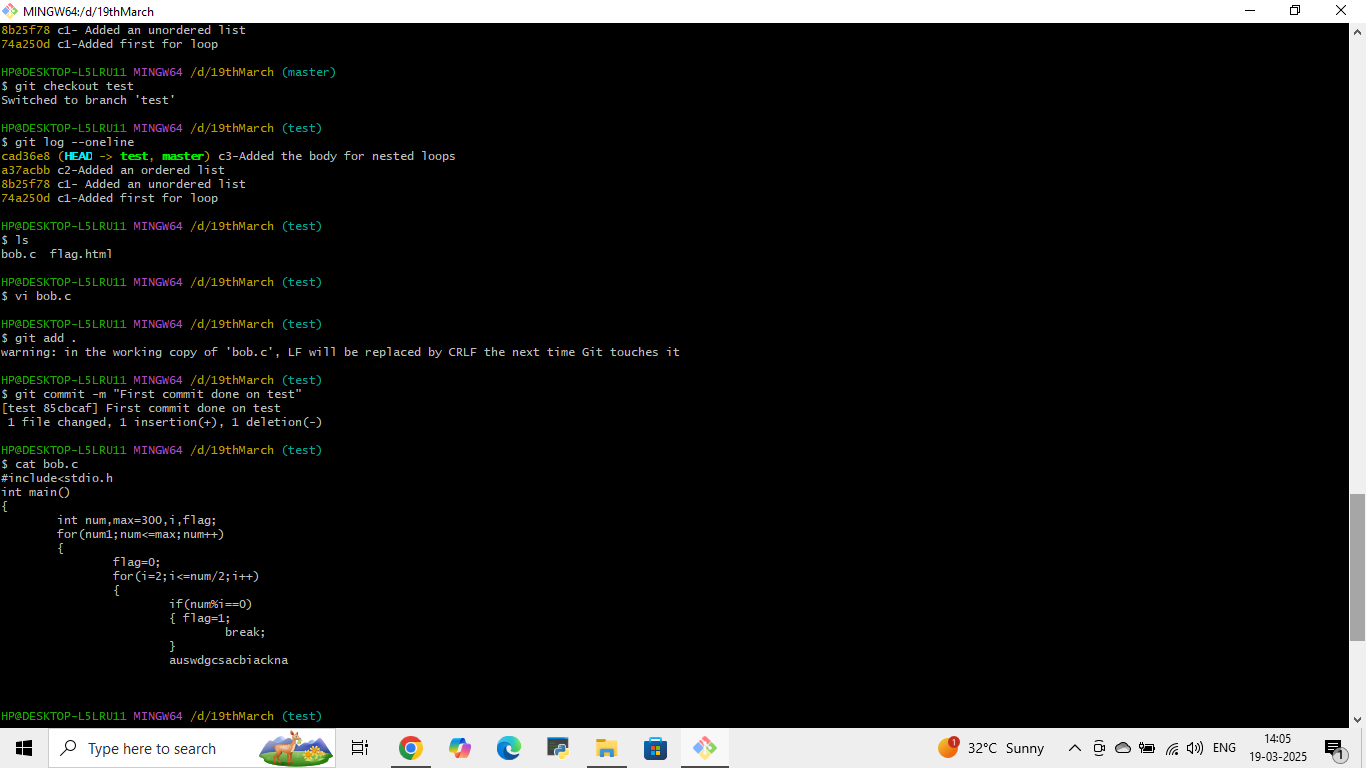
* **Code Review**: Ensures quality before merging.
* **Collaboration**: Discuss changes before they go live.
* **CI/CD Integration**: Automated tests run on PRs.

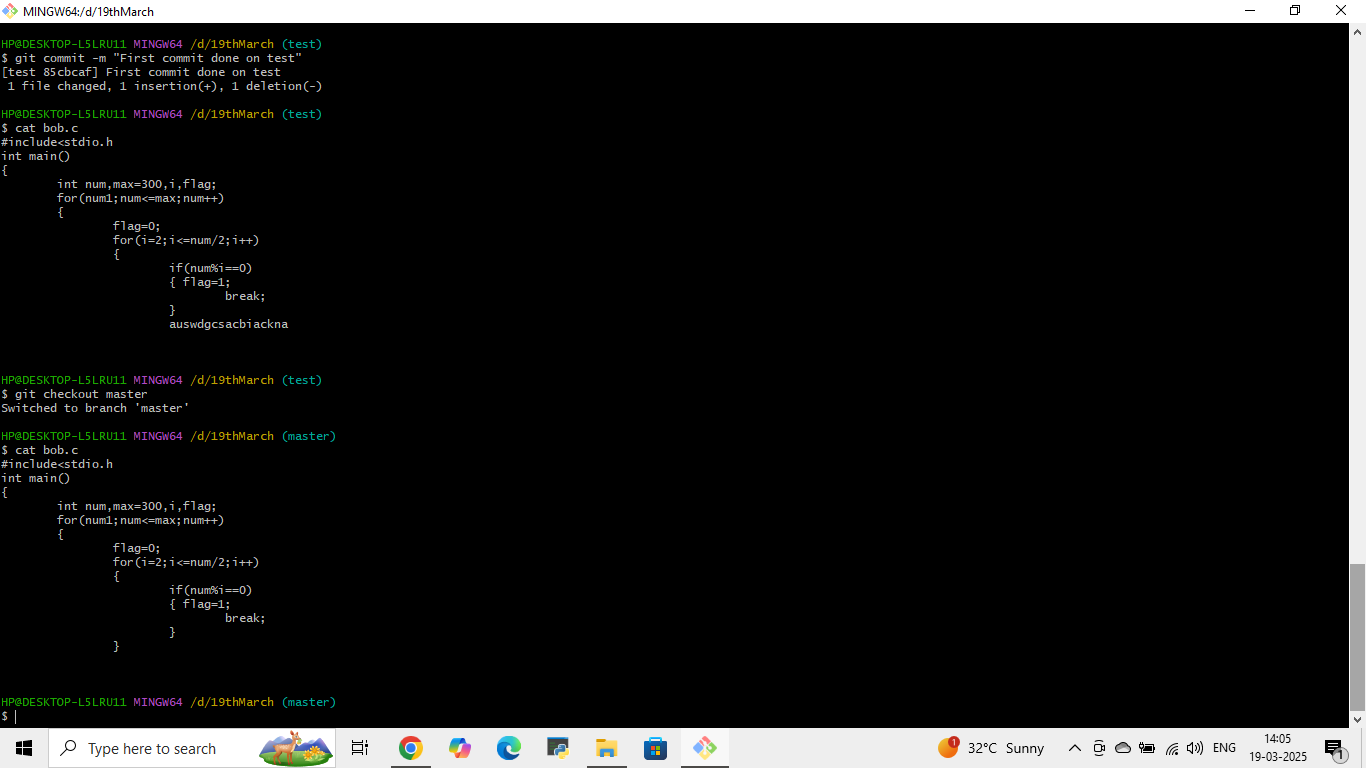
**Complete Git Process**

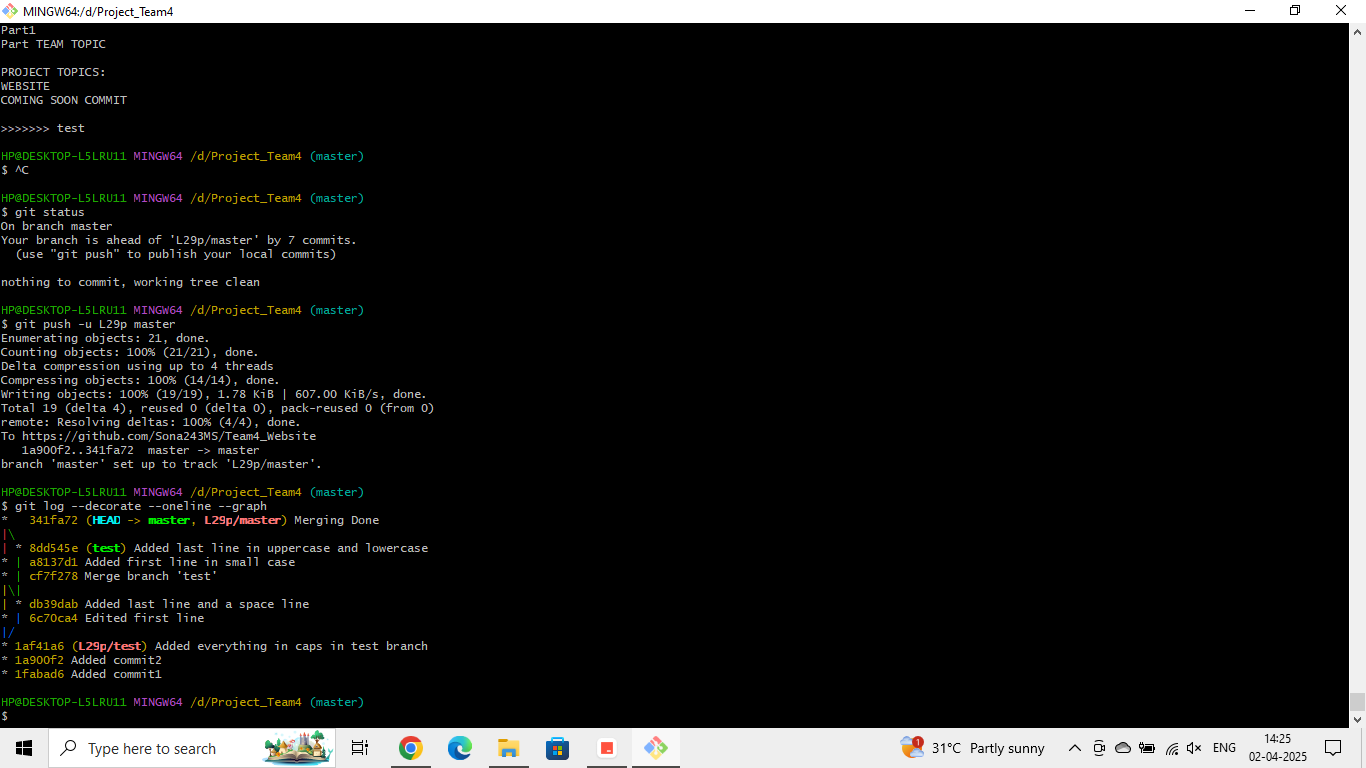
****

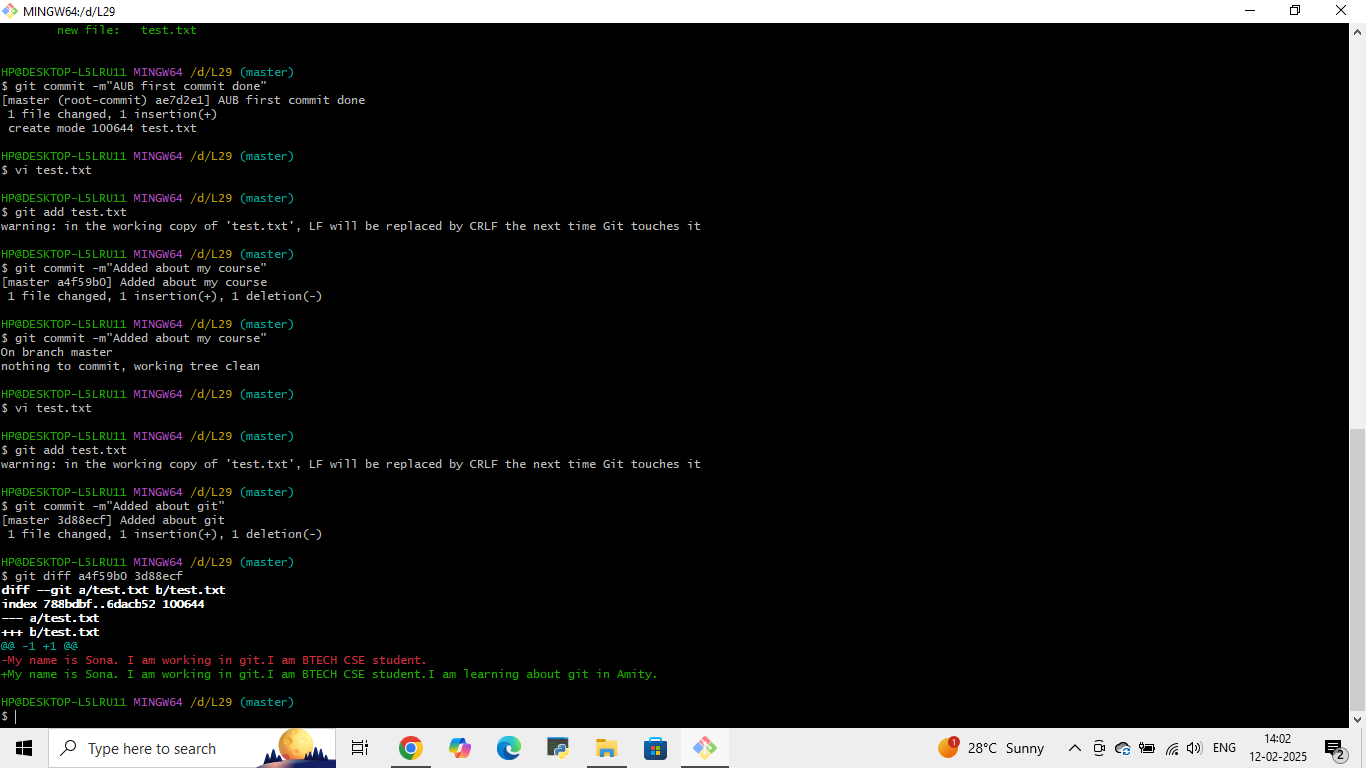
****

****

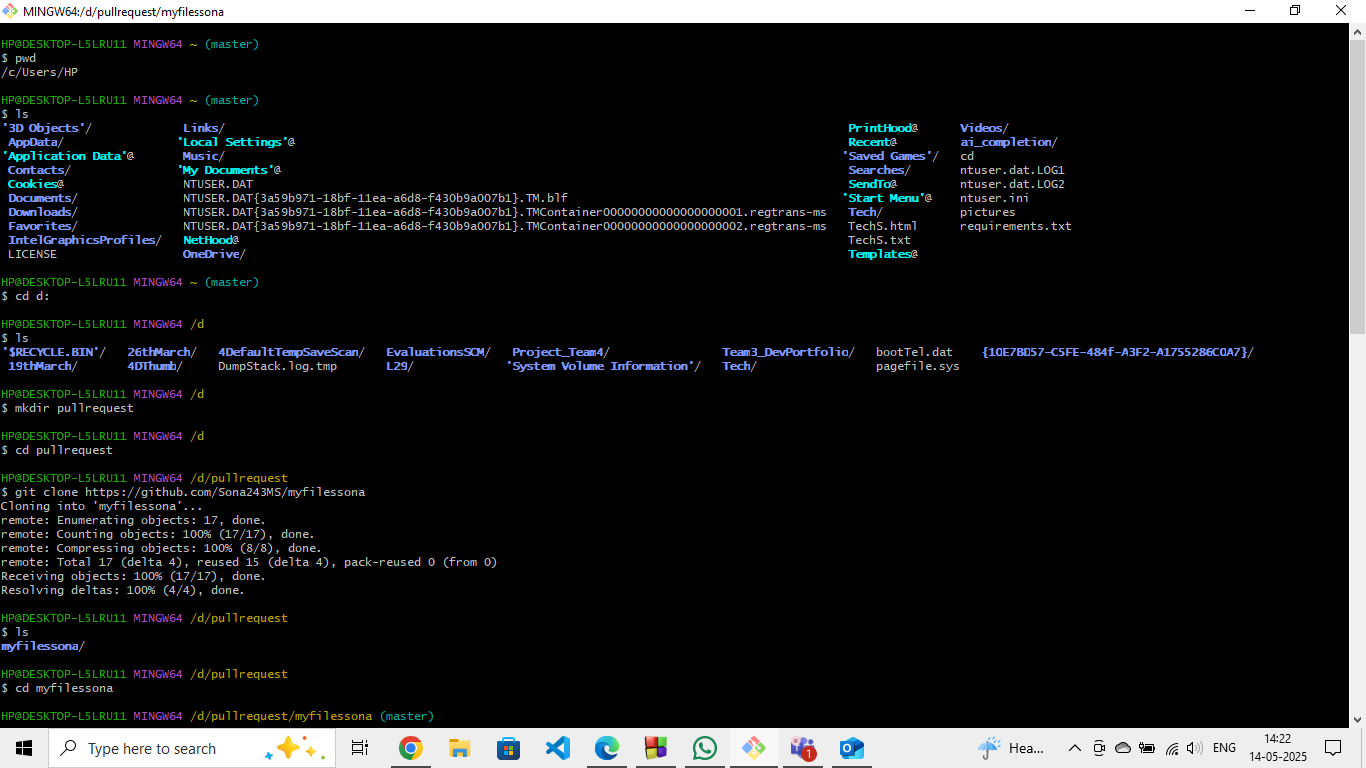
****

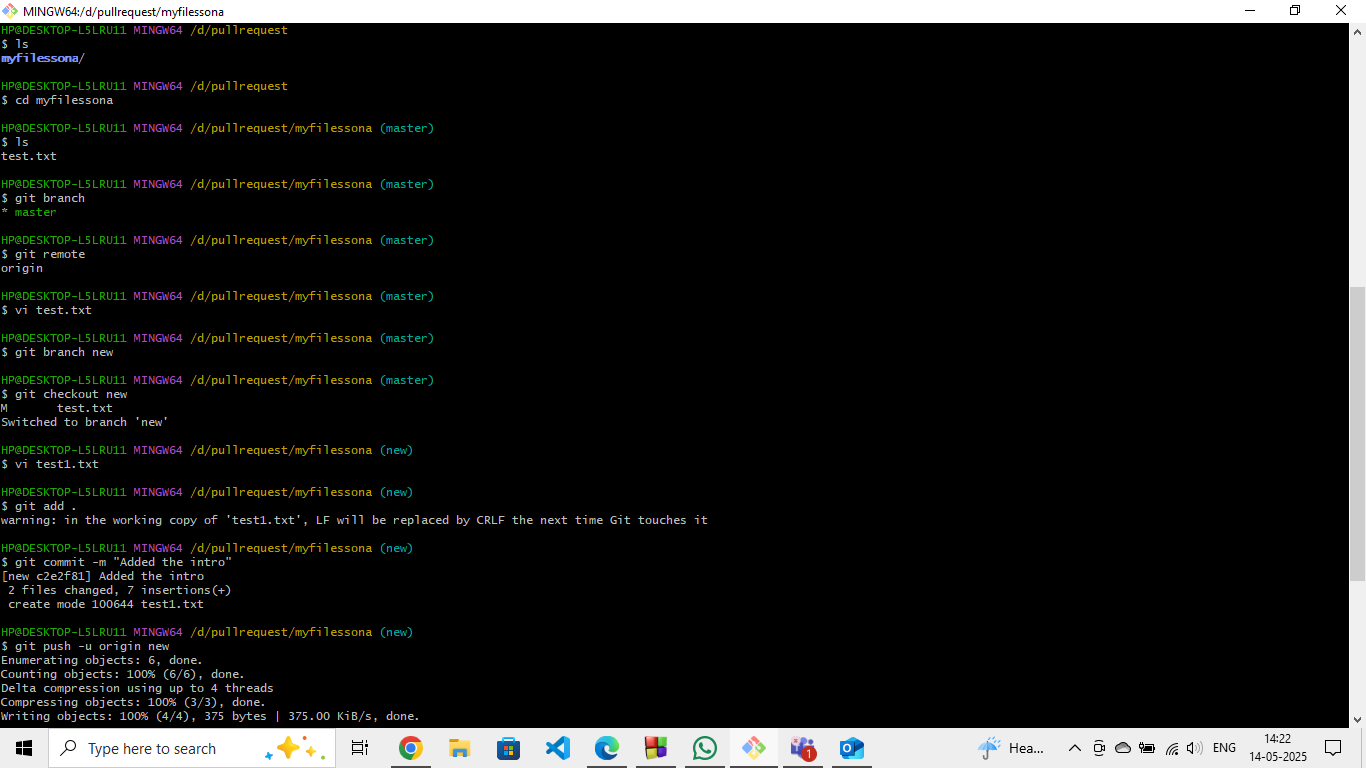
****

****

****

**.**





**CONCLUSION**

**The Power of Git in Modern Development**

Git is an **essential tool** for developers, offering **version control, collaboration, and workflow management**. Here’s a quick recap of its key benefits:

**1. Version Control & History**

* Track changes with commits
* Revert mistakes easily
* Maintain a clear history of project evolution

**2. Branching & Merging**

* Work on features independently with branches
* Merge changes safely with conflict resolution
* Support for **non-linear development** (feature branches, hotfixes)

**3. Collaboration & Remote Repositories**

* Share code via **GitHub, GitLab, or Bitbucket**
* Use **pull requests** for code reviews
* Enable **team workflows** (forking, cloning, pushing)

**4. Flexibility & Efficiency**

* Works **offline** (local commits)
* Supports **automation** (CI/CD pipelines)
* Cross-platform (Windows, macOS, Linux)

**Git Repository Link - https://github.com/Sona243MS**

**BIBLIOGRAPHY**

**Primary Source (Pro Git Book)**

1. **Chacon, S., & Straub, B.** (2014). *Pro Git* (2nd ed.). Apress.
   * **ISBN:** 978-1-4842-0076-6
   * **Online Edition:** <https://git-scm.com/book/en/v2>
   * **Key Topics:** Git fundamentals, branching, workflows, internals.

**Supplementary References (GitHub/Google)**

1. **GitHub Docs.** (2023). *GitHub Guides and Documentation*.
   * **URL:** [https://docs.github.com](https://docs.github.com/)
   * **Relevance:** Official guides on Git workflows, pull requests, and CI/CD.
2. **Google Developers.** (2023). *Git and Version Control*.
   * **URL:** [https://developers.google.com](https://developers.google.com/)
   * **Relevance:** Best practices for collaborative development.
3. **Stack Overflow.** (2023). *Git Questions and Solutions*.
   * **URL:** <https://stackoverflow.com/questions/tagged/git>
   * **Relevance:** Community-driven troubleshooting.