**SOURCE CODE MANAGEMENT**

**Course Code: CSE 2015**

**Slot: L14 L15**



**Name: Nikhilgouda Policepatil**

**SEN No.: A86605224105**

**Faculty: Dr Monit Kapoor**

**INDEX**

**1ST Lab Session:**

**Basics of Source Code Management**

**&**

**Git Installation**

**Computer**

A **computer** is any device that can carry out calculations, including both logical

and mathematical operations.

**Code/Program**

A **program (or code)** is a collection of instructions, typically arranged as an algorithm, that guides a computer to complete a specific task.

**Importance of Source Code Management**

Present-Day applications, like Spotify and Instagram, are complex systems consisting of multiple programs that interact seamlessly on both the frontend and backend to provide a smooth and engaging user experience. To maintain this level of performance and reliability, regular updates are vital for several key reasons:

* Bug Fixes: Quickly identifying and addressing errors that may arise, ensuring the application runs smoothly without interruptions.
* UI/UX Enhancements: Continuously refining the user interface and experience, making the app more intuitive, user-friendly, and visually appealing.
* Performance Optimization: Addressing and fine-tuning underlying issues that affect the app’s speed, responsiveness, and resource usage, resulting in a more efficient and reliable product.

For developers, effective management of source code is indispensable for several reasons:

* Preserving Context: Proper source code management ensures that all files and updates stay organized and in context throughout the entire development lifecycle, reducing confusion and errors.
* Facilitating Collaboration: It enables seamless collaboration among developers by allowing multiple people to work on a shared codebase without conflict, improving productivity and reducing the risk of overwriting work.
* Version Control: A well-managed system ensures that every change is tracked, allowing for easy rollback to previous versions if necessary, and helping developers understand the history of the project.
* Scalability and Maintenance: A structured approach to source code management ensures that as the application grows, it remains scalable and maintainable, avoiding technical debt and enabling smoother updates and feature additions.

In today's fast-paced development environment, effective source code management is not just a best practice—it’s a necessity for delivering high-quality software on time and maintaining long-term product success.

**Source Code Management Tools**

 **Git**:  
 A local version control system that allows you to monitor modifications and manage different versions of your project, ensuring efficient tracking of changes over time.

 **GitHub**:  
 An online platform that stores Git repositories in the cloud, allowing developers to collaborate, share code, and contribute to projects remotely from anywhere around the globe.

**Version**

In technical terms, a **"version"** refers to a specific iteration or state of a software program, document, or product, which represents a distinct point in its development, usually with updates, bug fixes, or new features compared to previous versions.

**Lab Practical 1**

* 1. **Installing Git**

**Step1.** Go to the pro git Book website and click on “1.5 Installing Git” under

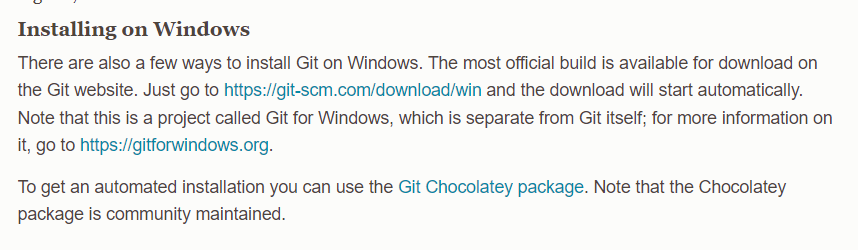
“Getting Started”.



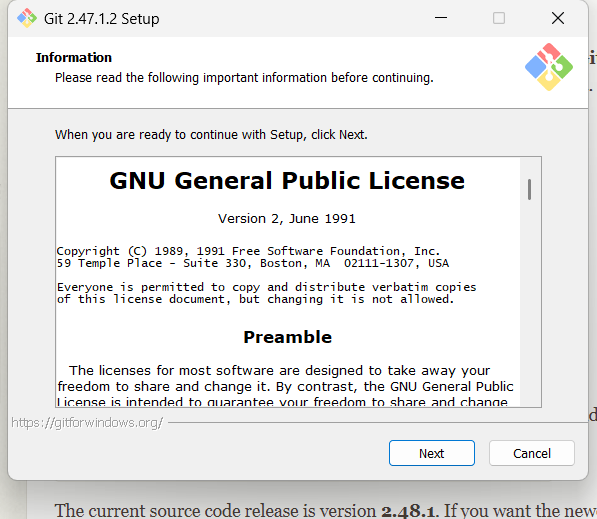
**Step2.** Scroll down to the passage titled “Installing on Windows” and click on

the first site in the passage.

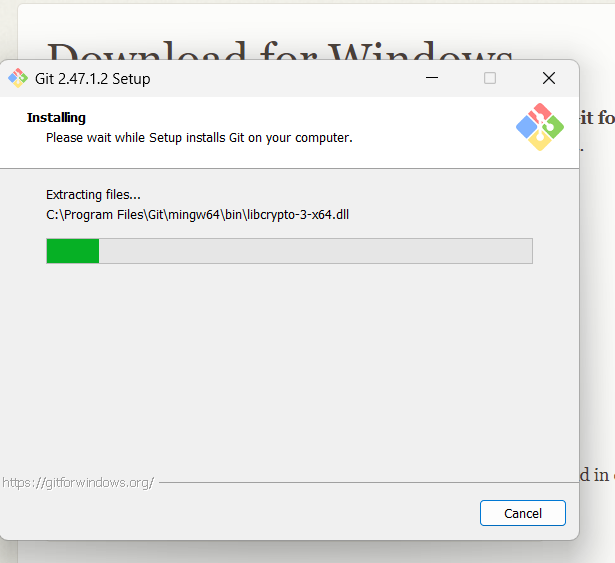
(The site should be:  <https://git-scm.com/download/win> )

****

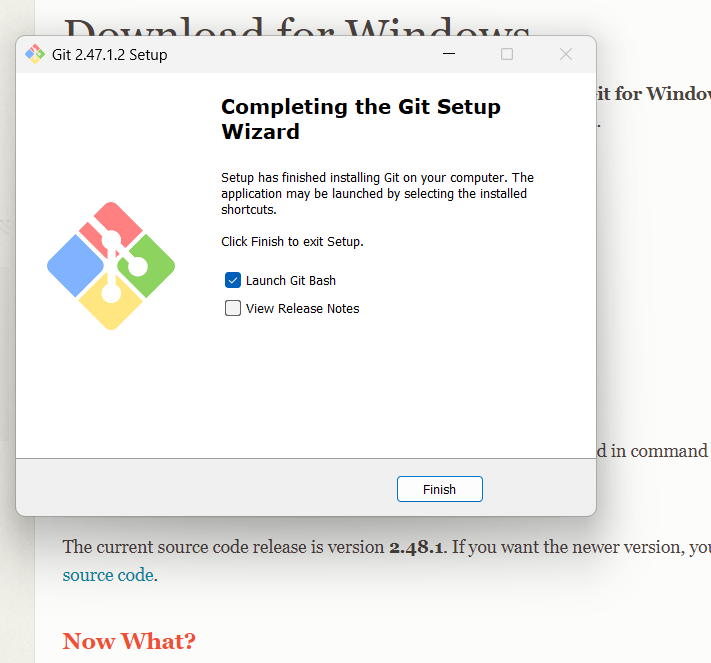
**Step3.** After clicking on the site, you will be directed to another page titled “Download for Windows”. Under “Standalone Installer”, click on “64-bit Git for Windows Setup” and it will start downloading for you, and you should see a pop-up window as shown below.



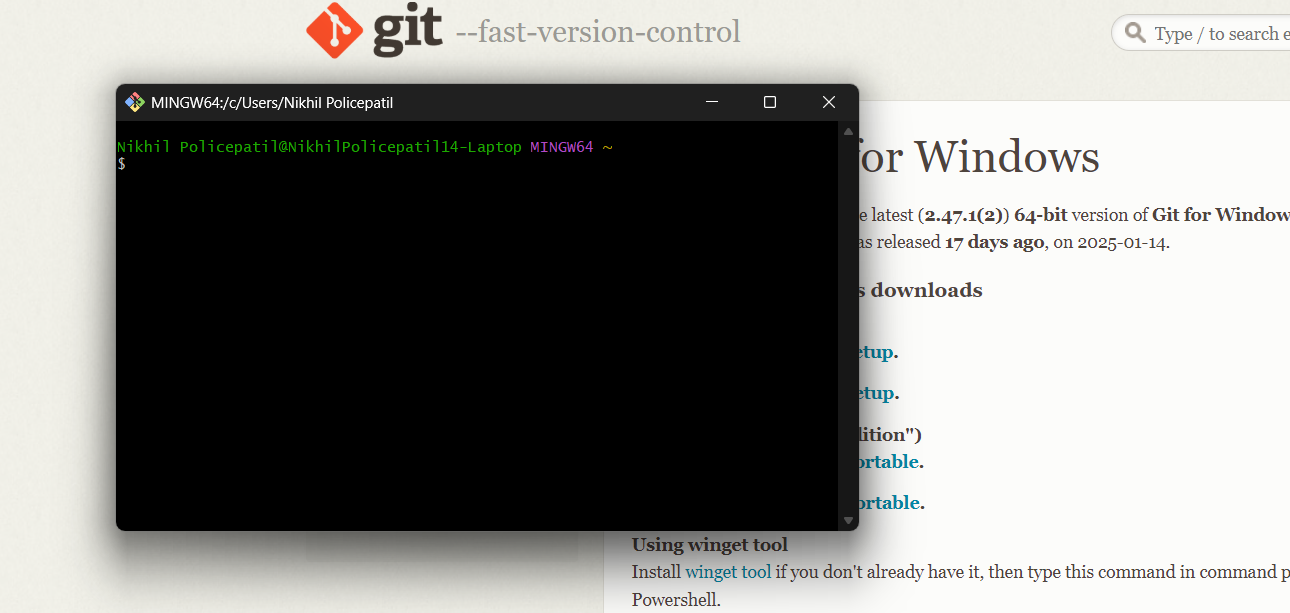
**Step4.** Click on “Next” on all the proceeding windows, and it will start Installing.



**Step5.** After installing, a Git window will show up, giving you a choice on Launching Git Bash. Click on “Launch Git Bash” and then click on “Finish”.



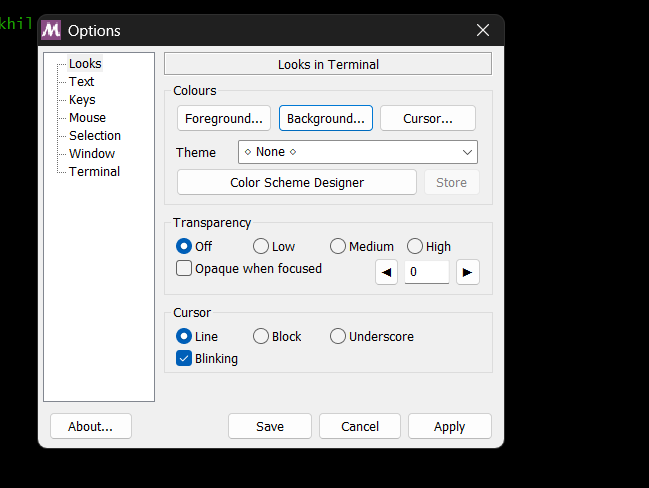
**Step6.** Once you have done all the 5 steps above, a Git Bash window will show up as shown below.

****

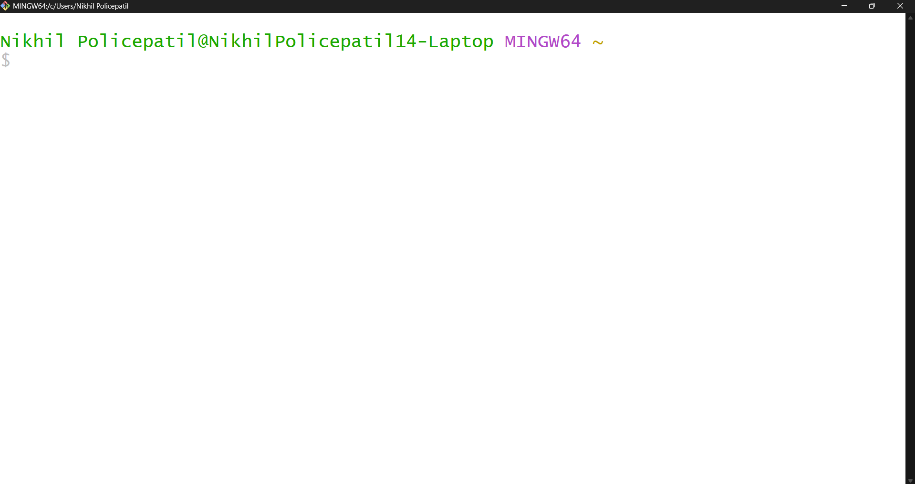
* 1. **Customization of Git Bash**

**Step1.** Right click on the Git Bash window, and click on options.

**Step2.** After clicking on options, an options window should show up.

****

**Step3.** Click on “Background” and pick any colour for the background. You should see the colour change after applying and saving.

****

* 1. **Basic CLI Commands**

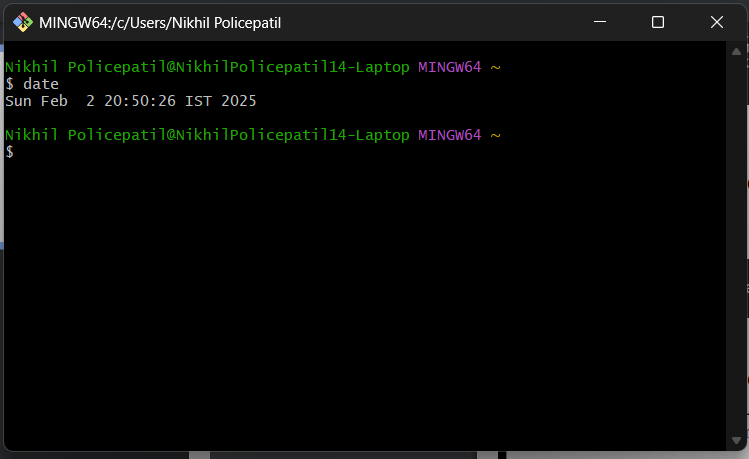
1. **Command: ls**

**Description:** Lists all files and directories in the current directory.



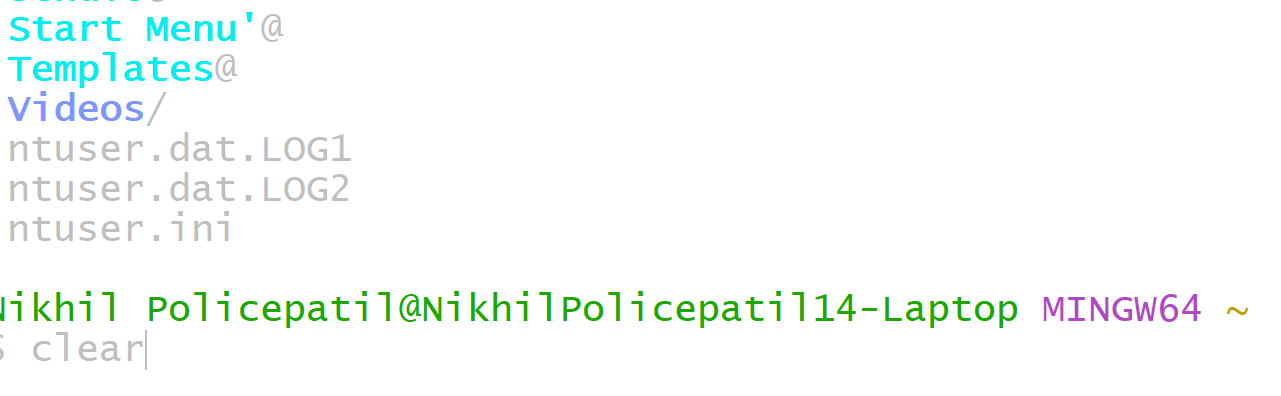
1. **Command: date**

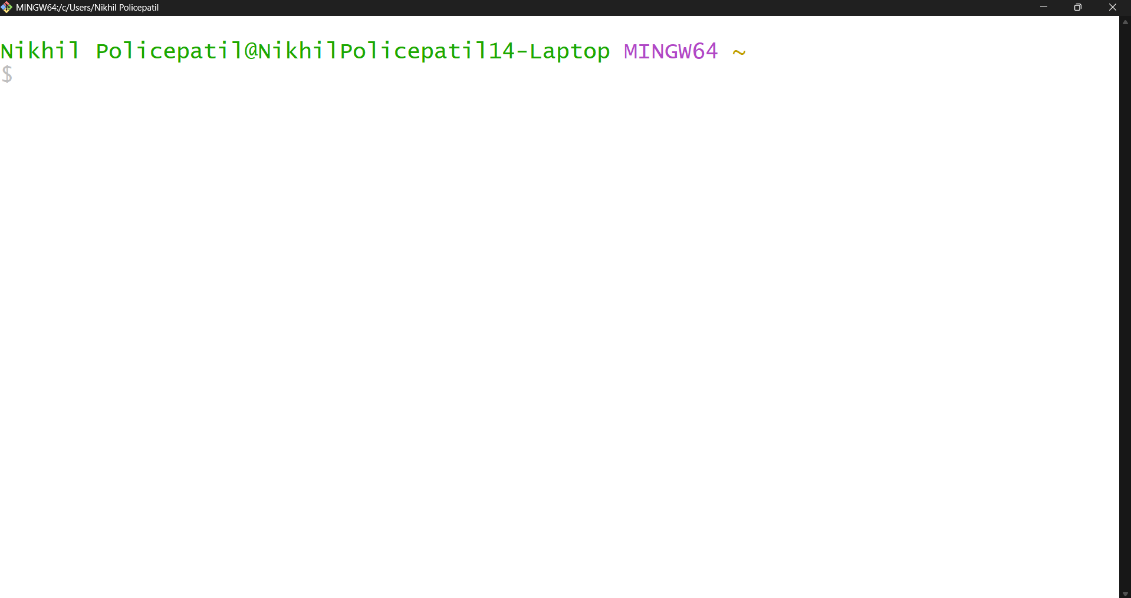
**Description:** shows the current date and time in a standard format



1. **Command: clear**

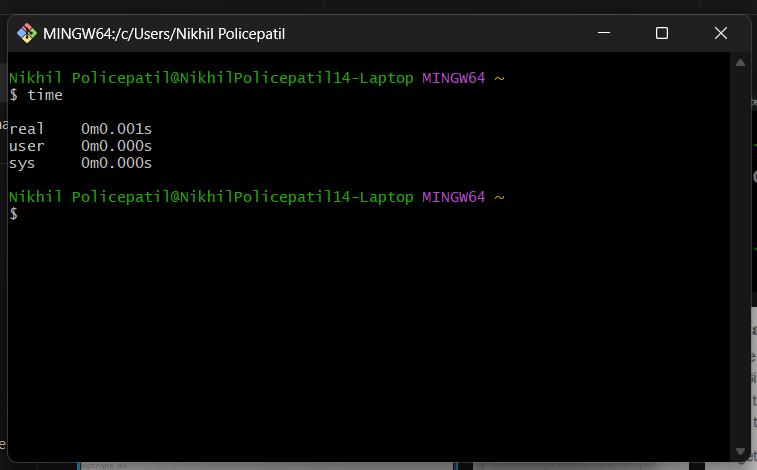
**Description:** The clear command in the CLI is used to clear all the current text and output displayed in the terminal window.





1. **Command: time**

**Description:** The time command in the CLI is used to measure the execution time of a command or program.

****

1. **Command: rm hello.txt**

**Description:** Removes the file hello.txt from the current directory.

1. **Command: cat hello.txt**

**Description:** The cat command (short for concatenate) is used to display the contents of a file.

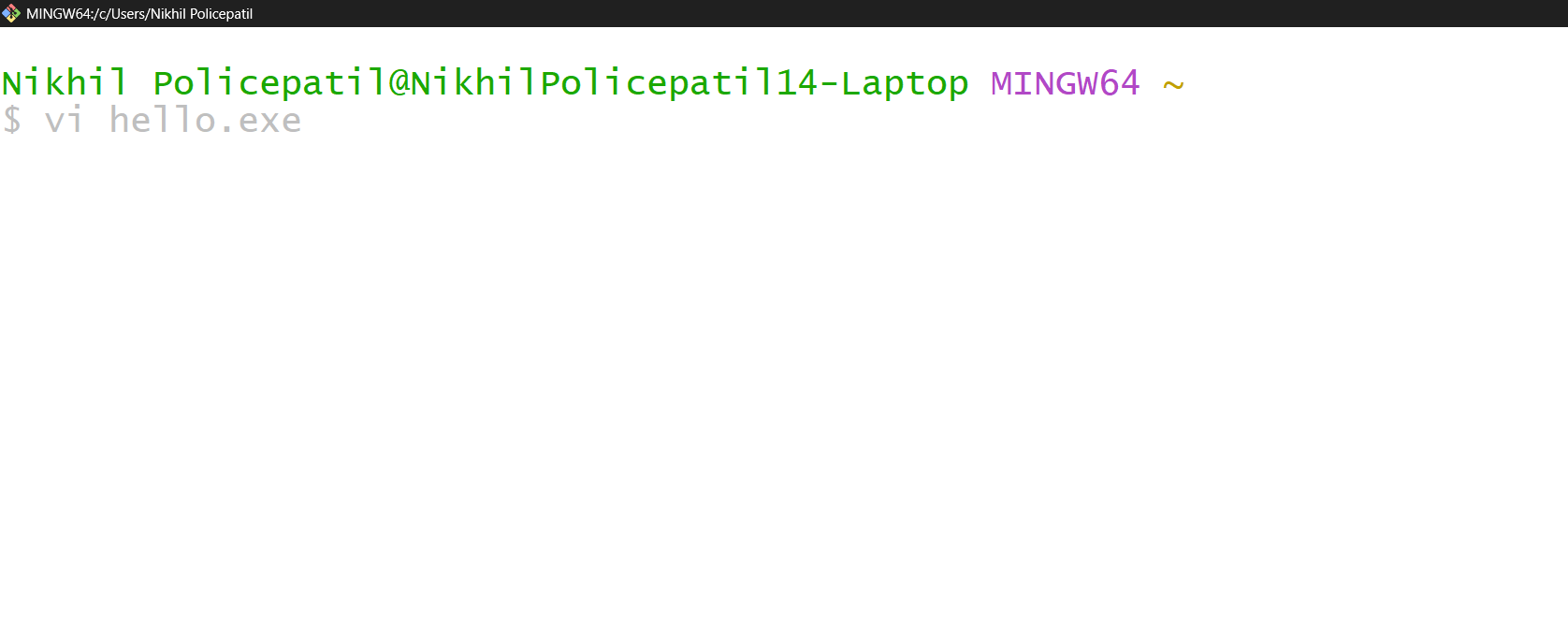
****

1. **Command: cd Desktop**

**Description:** Changes the current working directory to the Desktop directory.

* 1. **Vim Text Editor** 
     1. **Command: vi hello.txt**

**Description:** Opens (or creates) the file hello.txt in the Vim text editor.

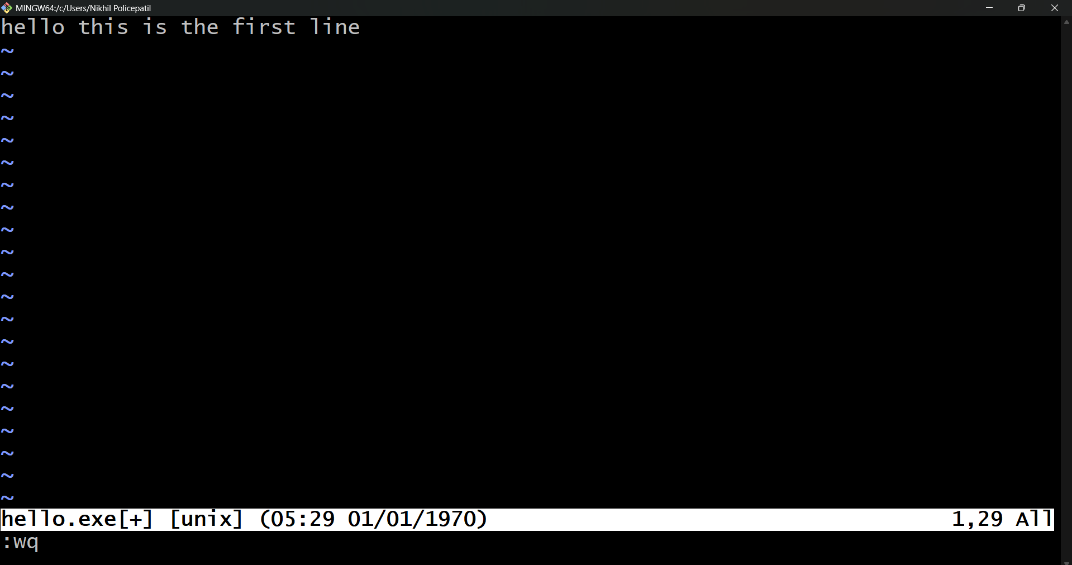
****



* + 1. **Command: i (Insert Mode)**

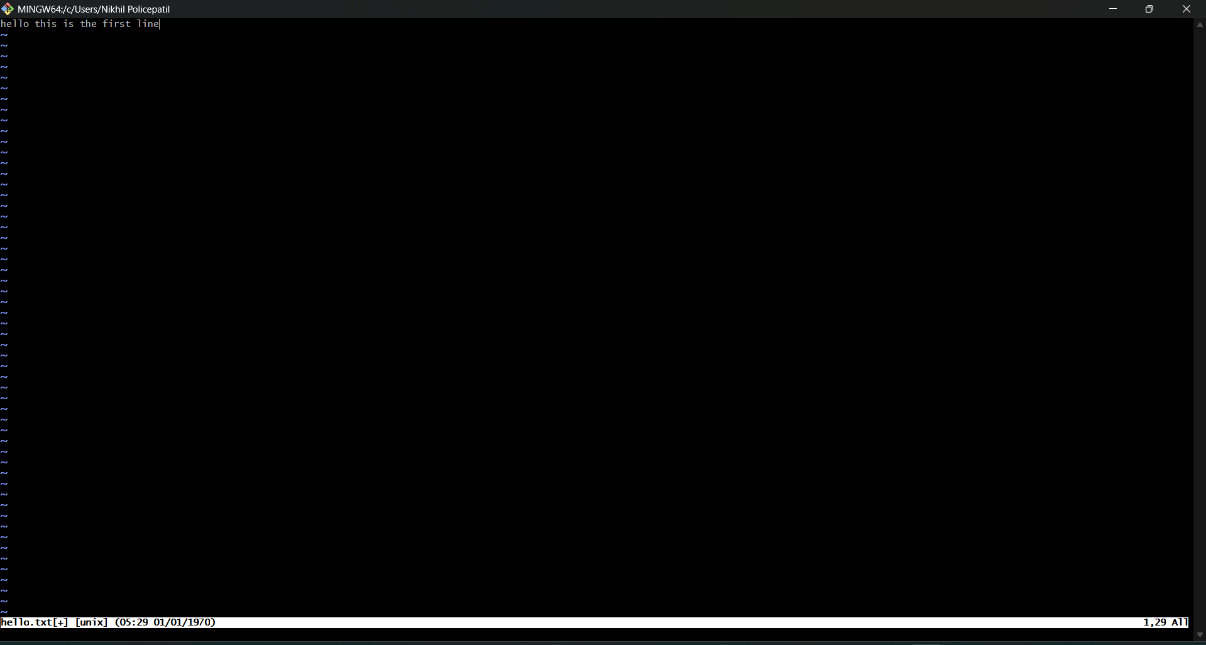
**Description:** Enters insert mode in Vim to allow text input.

****

****

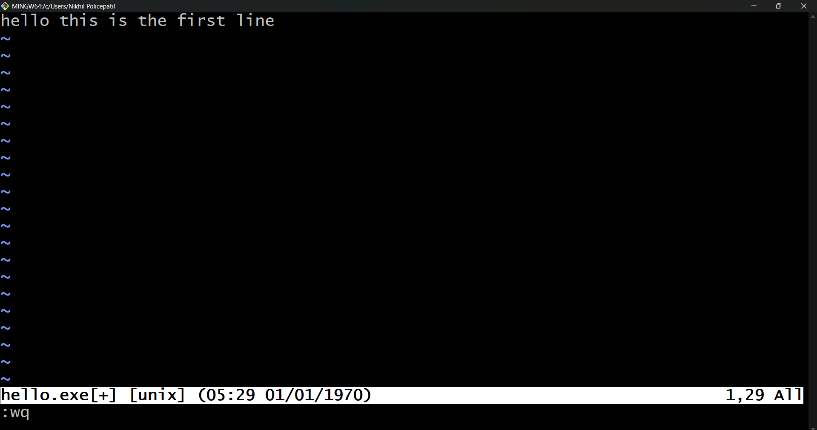
* + 1. **Command: esc**

**Description:** Used to exit insert mode

****

* + 1. **Command: :wq**

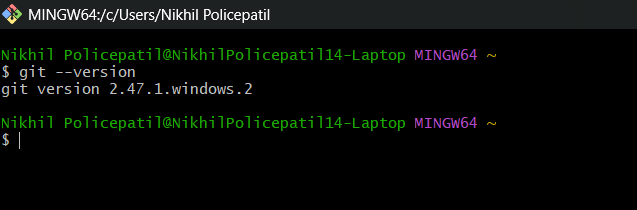
**Description:** Saves the changes and exits the Vim editor.

****

****

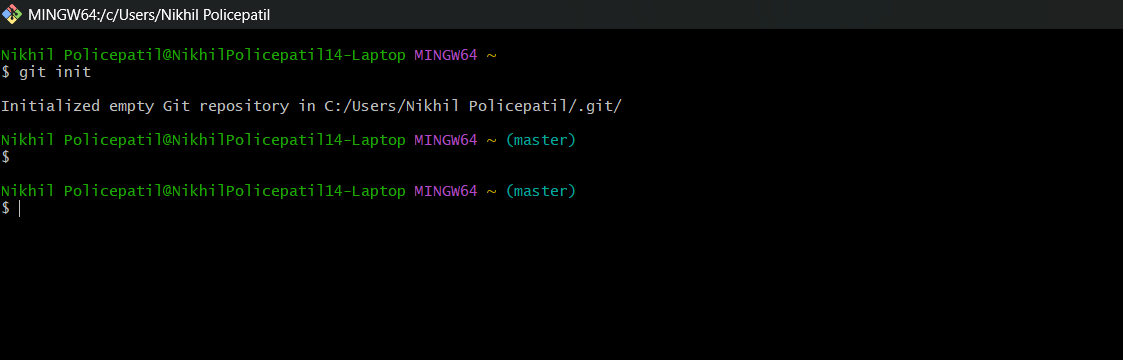
* 1. **Git Commands**
     1. **Command: git - - version**

**Description:** The ”git –version” command is used to check the installed version of Git on your system.



* + 1. **Command: git init**

**Description:** Initializes a new Git repository in the current directory**.**

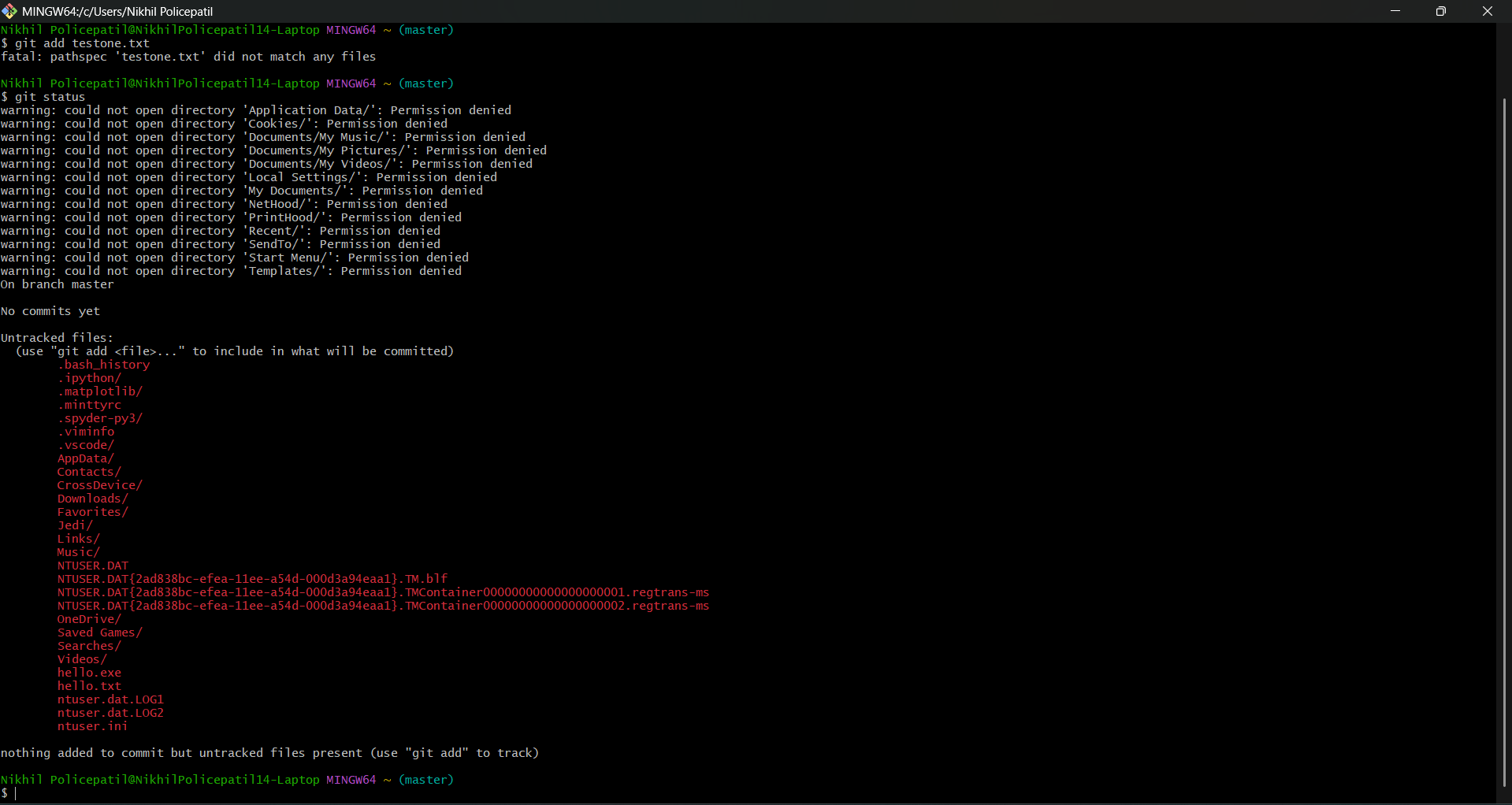
****

* + 1. **Command: git status**

**Description:** Displays the current status of the working directory and staging area.

* + 1. **Command: git add testone.txt**

**Description:** Adds testone.txt to the staging area in preparation for a commit.

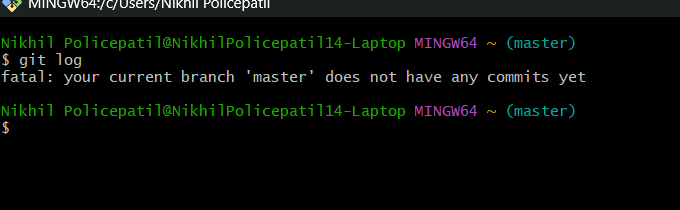
****

* + 1. **Command: git commit -m "add file one"**

**Description:** Commits the staged changes with the message "add file one”

* + 1. **Command: git log**

**Description:** Displays the commit history of the repository.

****

**Lab Session 3: Git Diff**

**Mount Point:**

Point from where we can access the desired folder directly

Documents/com~apple~CloudDocs/Amity/Git\_Amity/Lab\_2 In the above file path mount point of Lab\_2 is Git\_Amity Lets make modifications in our repository before demonstrating git diff

**Command touch:**

Used to create a file without any content

**Command git rm -rf (filename):**

Used to remove a file from git tracking

**Task 1: make two commits in a directory**

**Step 1: create two files with content in a directory**

**Step 2: use git add. Command to add both files in the staging area**

**Step 3: use git commit -m to commit both files to local repository**

**Step 4: use git log command to verify the commit history.**

**Step 5: use git log- - oneline for generating shorter commit id**

**Task 2: compare two commits in a directory**

**Use git diff along with the commit id generated from git log - - online**

**Lab Session 5 : Working with remotes**

**Step1: Make 4 commits and compare them using git dif**

**Step 2: Use git remote command to establish a connection between local Git repository and a remote repository**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**Step 3: Use The git push command is used to upload local repository content to a remote repository**

**Step 4: Confirming the remote connection with git remote**

**Step 5: Checking the commits made on GitHub account**

**Git pull is used to fetch and integrate changes which are in the remote repository to local repository**

**Git remote-v: used to view all remote repositories in a directory lists all configured remote repositories along with their corresponding URL**

**Lab Session 6 : git branching**

**Branch:** pointer to a commit

**Pointer:** connects two memory address where at least one variable must have an active memory address

**Head:** branch on which the last commit is made

**Git branch command used to view the existing branches in the git repository**

**Git checkout command used to switch the currently active branch to another branch. Here we want to create a new branch from a particular commit**

**Git branch test\_1:** used to create branch with name test\_1 Confirm the created branch by using git branch command to view all the branches

**Use git\_checkout command to pivot to that particular branch**

**Make commits in test\_1 branch**

**Viewing the commits on a particular branch**

**Merging Branches**

Step 1: committing changes in hello .txt on main branch

Step 2: Creating test branch

Step 3: Switching to test branch

Step 4: making changes in hello.txt in test branch

Step 5: committing the changes

Step 6: merging the test branch

**Lab Exercise 7: Merge Conflicts**

Below screenshot reflects the current configuration of our repository

Step 1: create another test branch and switch to that branch

Step2: modify hello.txt and commit those changes

Step 3: merge test branch to test-2 branch. A merge conflict will appear

Step 4: Resolve the merge conflict using git merge tool

Step 5: commit the changes

Step 6: checking hello.txt for confirming the merge

Step 7: using git log - - graph - - decorate for visual representation of branches.

**Lab Exercise 8: Fork Clone Workflow and Sending Pull request**

Step1: Navigate to a desired repository and fork it

Step 2: using git clone command create a copy of the repository in the local system

step 3: create your own feature branch for making the required changes

Step 4: commit those changes

Step 5 : push the changes to forked repo

step 5: navigate to GitHub and send pull request to dev/test branch

Step 6: navigate to insights and then to network graph to see the overall branching workflow

**Lab Exercise 9: Accepting pull Requests**

Step 1: . Navigate to pull request section of your GitHub repo

Step 2: Accept the pull request

Step 3: In your local git repository checkout to dev branch

Step 4: execute git pull to integrate the commit on GitHub to local system repo

Step 5: checkout main branch merge the the dev branch

Step 6: push the main branch to GitHub

Step 7: Confirm with network graph on Github

**Lab Exercise 10: .gitignore**

Step 1: added multiple files with different extensions

Step 2: initialised git repository and included.gitignore file

Step 3: staging area before committing gitingore file

Step 4: committing gitignore file

Step 5: staging area after committing gitignore file

**SCM PROJECT**



Name: Nikhilgouda Policepatil

Rollno.: A86605224105

Course: B.tech CSE

Slot: L-15-16

Submitted to: DR. Monit Kapoor

**AIM:**

The aim is to create a repo where in you will create a code of 600 lines each member in any coding language Using Git Bash and Git Hub.

**Requirements:**

* You have to make at least 3 branches
* There has to be at least 3 commits in each branch
* You have to make changes in your team members repos.
* The code should run successfully

**Full Project Report/Log:**

**Step 1:** I Made a file in git bash using “mkdir” and name the file, SCMProj.

After that change the directory to that file using the command cd.

A screen shot of a computer

AI-generated content may be incorrect.

**Step 2:** I made a git repository in git bash using git init, and make a repo on git hub.

A screen shot of a computer

AI-generated content may be incorrect.

**Step 3:** I then connected the file that I had created in git bash to the repo that I made in git hub by using the command git remote add origin “URL”

A screen shot of a computer code

AI-generated content may be incorrect.

**Step 4:** I then made 3 branches named “Addelements”, “design” and “final”,

Each separately.

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 5:** I Put my files that contain the code to my website into the first branch called design.

**Step 6:** I then made changes to my code in my files and then added them using “git add .”, then committed them using “git commit -m “name of commit” ”and finally pushed them to my git hub account using “git push -u origin design”.

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 7:** I then exited the branches once I was done with the commits using “git checkout master” and then merged the branches to my master branch using the command ‘git merge (branch name)” and then pushed the merges to my git acc using “git push -u origin master”

A black background with a black square

AI-generated content may be incorrect.

**Step8:** once I had merged all my branches that I had made commits to, to my master branch, I then started working on my team mates branch.

**Step 9**: I first went to my friends repo and created a fork and named the cloned repo

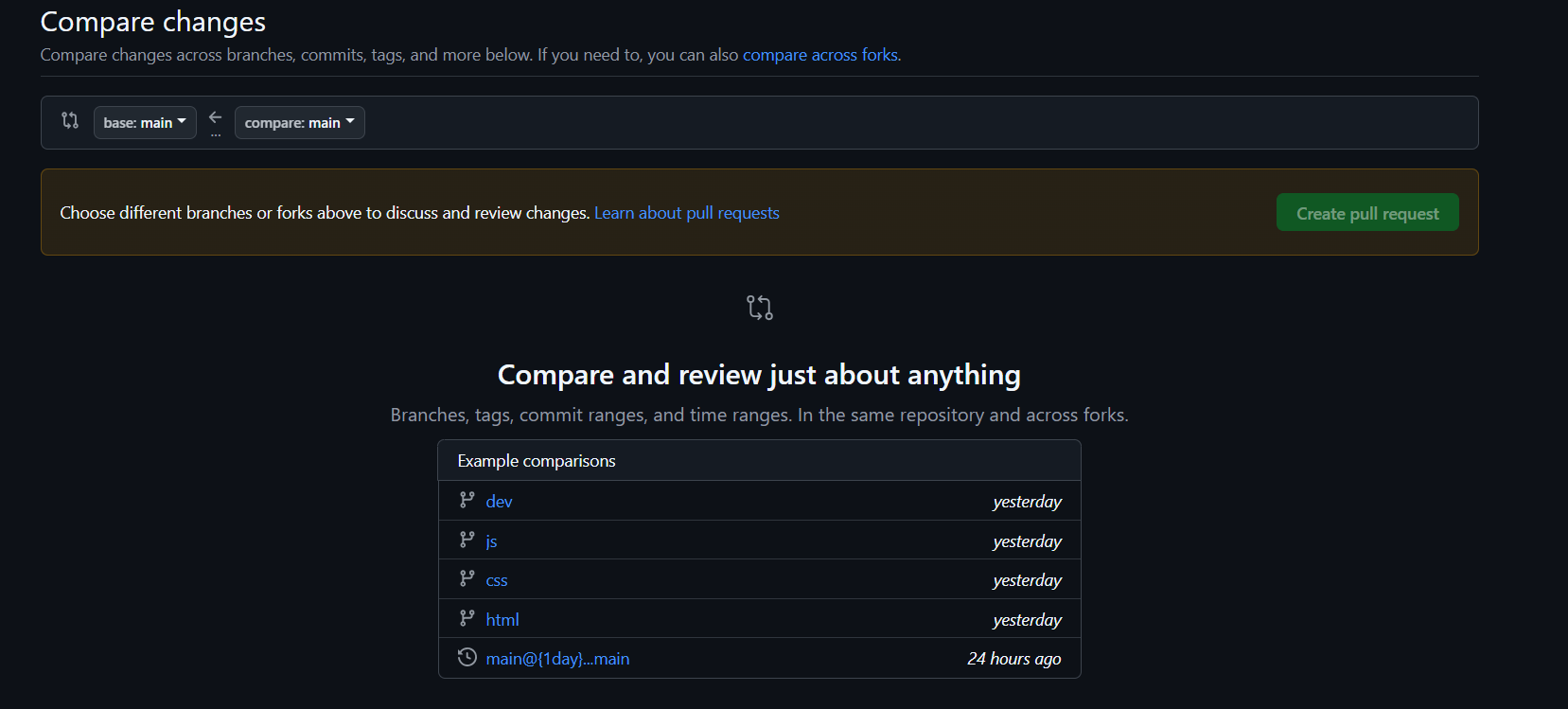
A screenshot of a computer

AI-generated content may be incorrect.

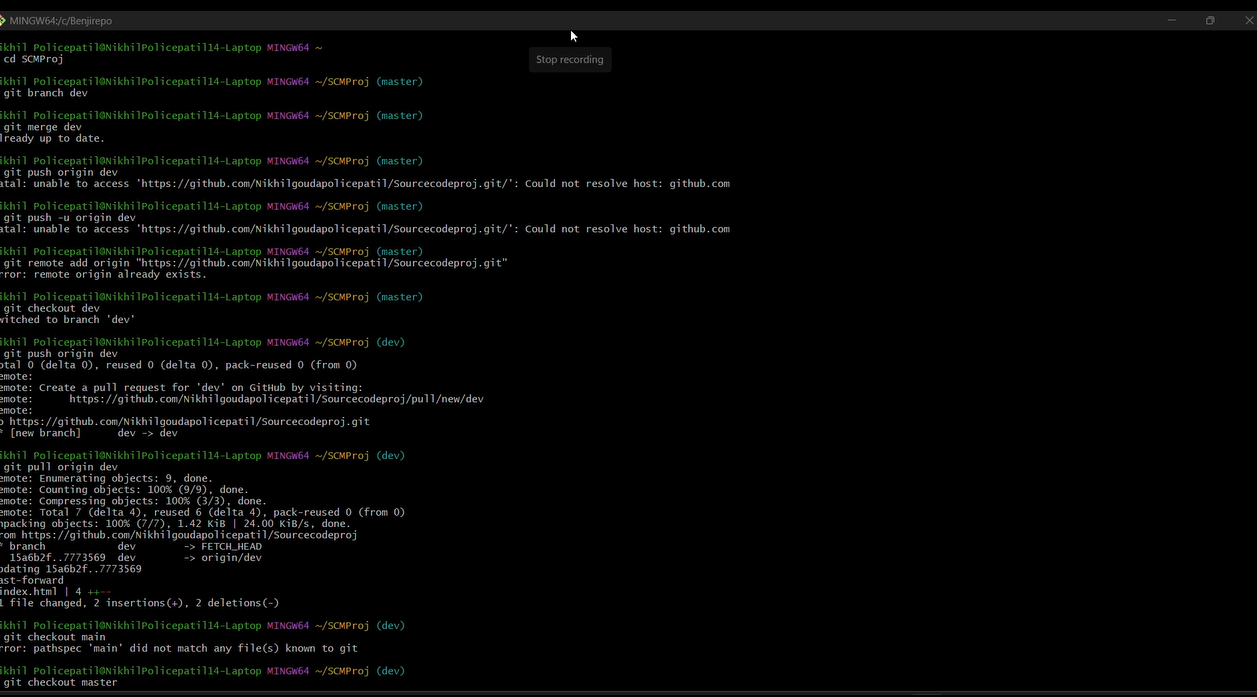
**Step 10:** I then went to my git bash and opened the cloned repo and made a branch called “Nikhilfix”, where I can then change his codes, and add and commit them like regular.



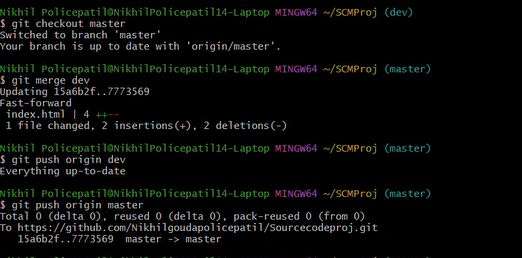
**Step 11:** After making changes to my team mates codes, I went to his repo in git hub and added a pull request and the rest is from his side.



**Step 12:** My team mate then sent a pull request to my repo where had change my codes and made it a little better, I then merged the pull request and went to git bash and opened my files which is linked to my own repo in git hub and made another branch called dev branch using the command “ git branch dev”



**Step 13:** I then pushed the dev branch to my repo in git hub and then pulled the files that were edited by my team mate and merged by me, and then I had went back to my master branch and merged the dev branch to my master branch and then pushed both the branched to my repo in git hub, thus the changed that my team mate had made where now integrated into my codes in my repo.



**Network Graph:**

As you can see, graph has a master branch and a dev branch from my team mate, the straight master branch shows that I had no conflicts while I was merging my branched and the that the merges where seamlessly integrated.

A screenshot of a computer

AI-generated content may be incorrect.

Our website:

A red circle with white text

AI-generated content may be incorrect.

A screenshot of a website

AI-generated content may be incorrect.

A screenshot of a phone

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a screen

AI-generated content may be incorrect.