STT Lab1

Name=Rapuru.Ganesh

Roll no=23110271

# Introduction, Setup, and Tools

### Overview

The present study investigated the foundations of Git, a distributed version control system commonly used in software development to monitor changes and foster cooperation. Implementing a continuous integration workflow with GitHub Actions and Pylint for code quality assurance, setting up Git, initialising a local repository, managing files through staging and committing, and connecting with remote repositories on GitHub were some of the responsibilities.

#### Objectives

* To understand the importance of **Version Control Systems (VCS)** in software development.
* To learn the basic **Git commands** such as configuring user details, initializing repositories, staging, committing, and viewing history.
* To practice creating and linking a **local repository with GitHub**, and perform operations like push, pull, and clone.
* To gain hands-on experience with **remote repositories** and collaborative workflows.
* To write a **Python program (≥30 lines)** and integrate it with GitHub.
* To set up a **GitHub Actions workflow with pylint** for automated code quality checks.
* To resolve pylint issues in the code until the workflow successfully passed with a green tick.

#### Environment Setup

* Set up Git on the local system.
* Create repositories and register for a GitHub account.
* Set up Git using your login information.
* Install Python3

### Tools and version used

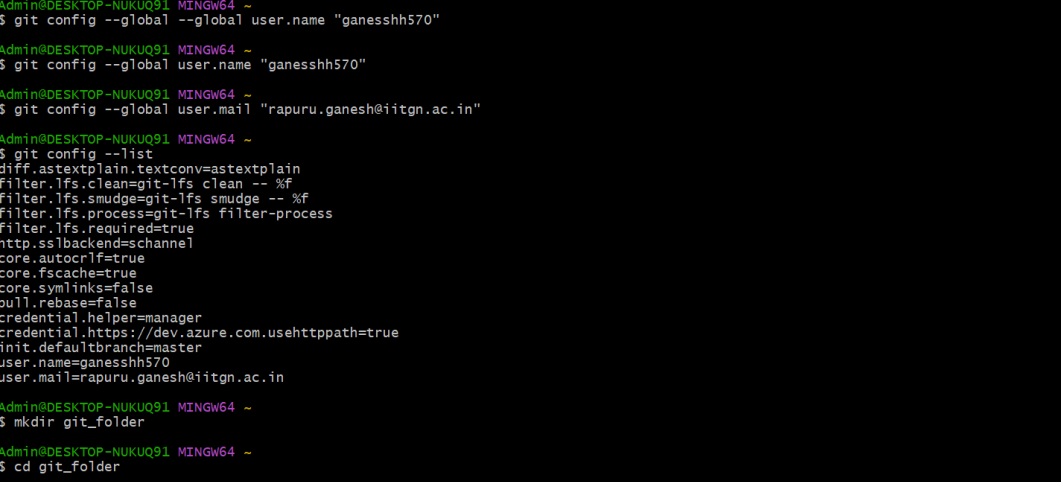
* Git: Version 2.30.1
* GitHub: Web-based platform for version control and collaboration.
* Python: Version 3.9.1
* pylint: Version 2.6.0
* GitHub Actions: Continuous integration service provided by GitHub.

# Methodology and Execution

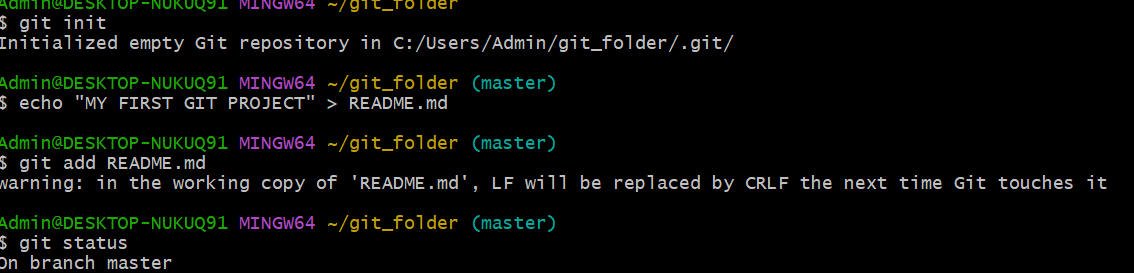
## Git Basics

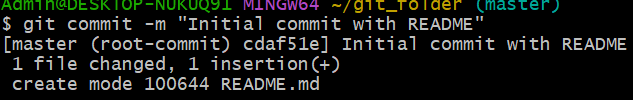
* Setting up Git:

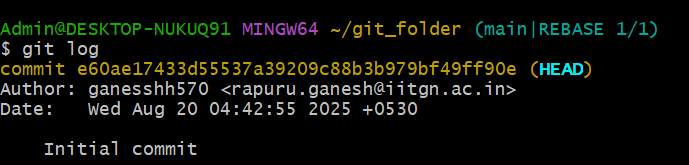




* Initializing a Local Repository:

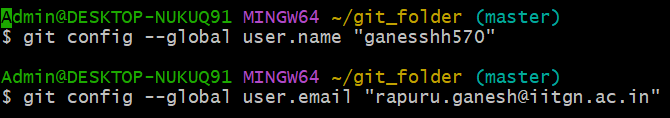


* Adding and Committing Files:
* Viewing the Commit History:

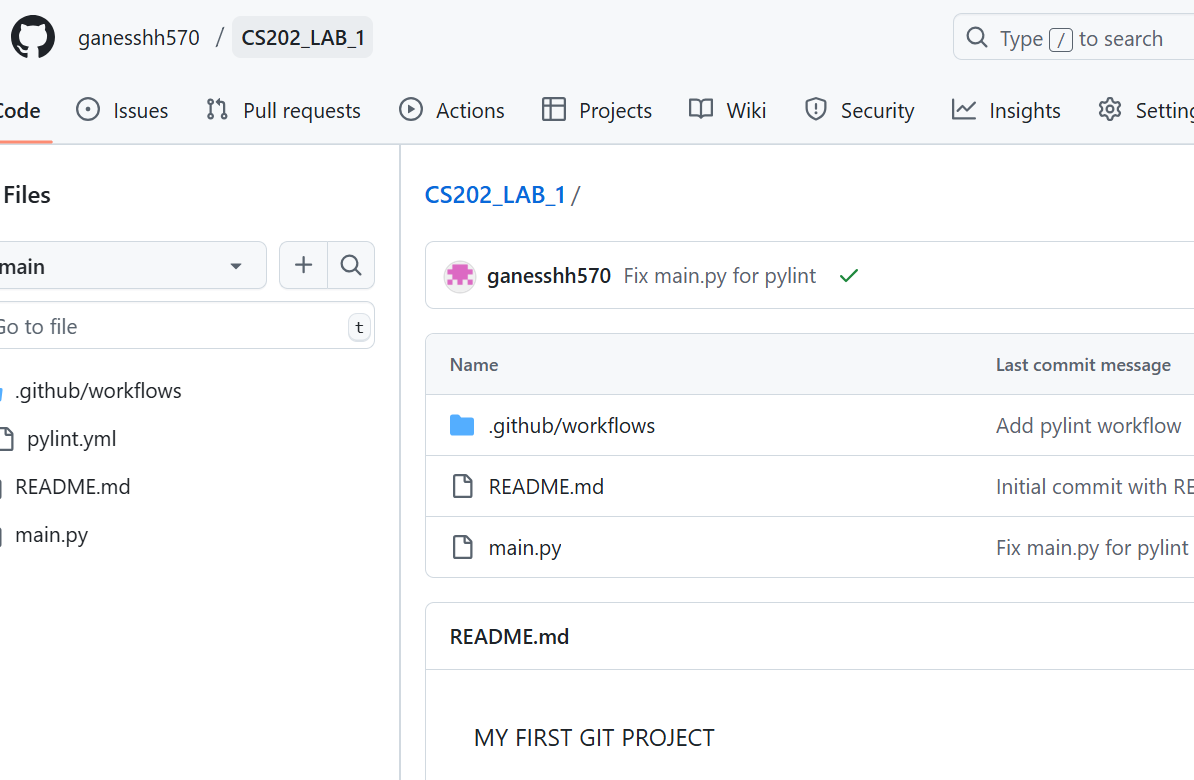


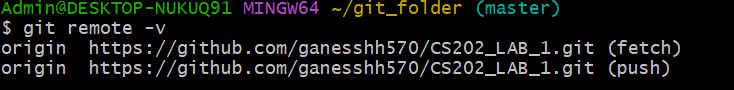
## Working with Remote Repositories

* Connecting to GitHub:

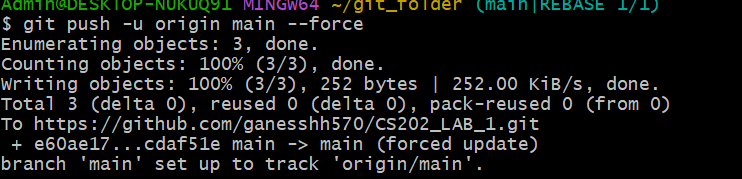


* Pushing Changes to GitHub:



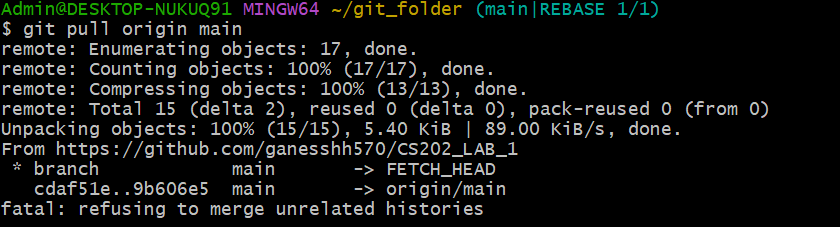


* Cloning a Repository:

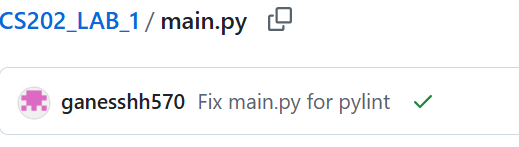


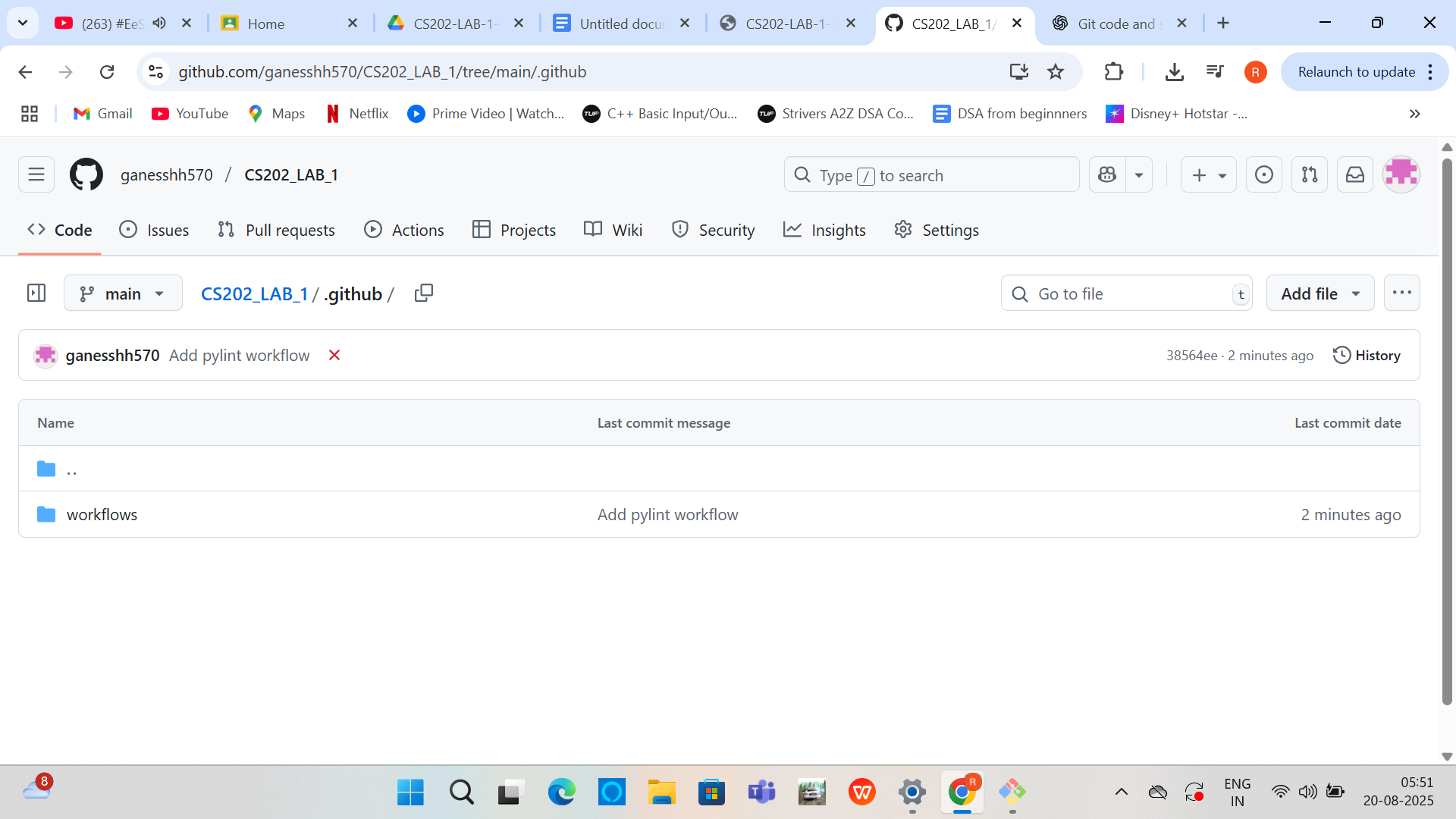


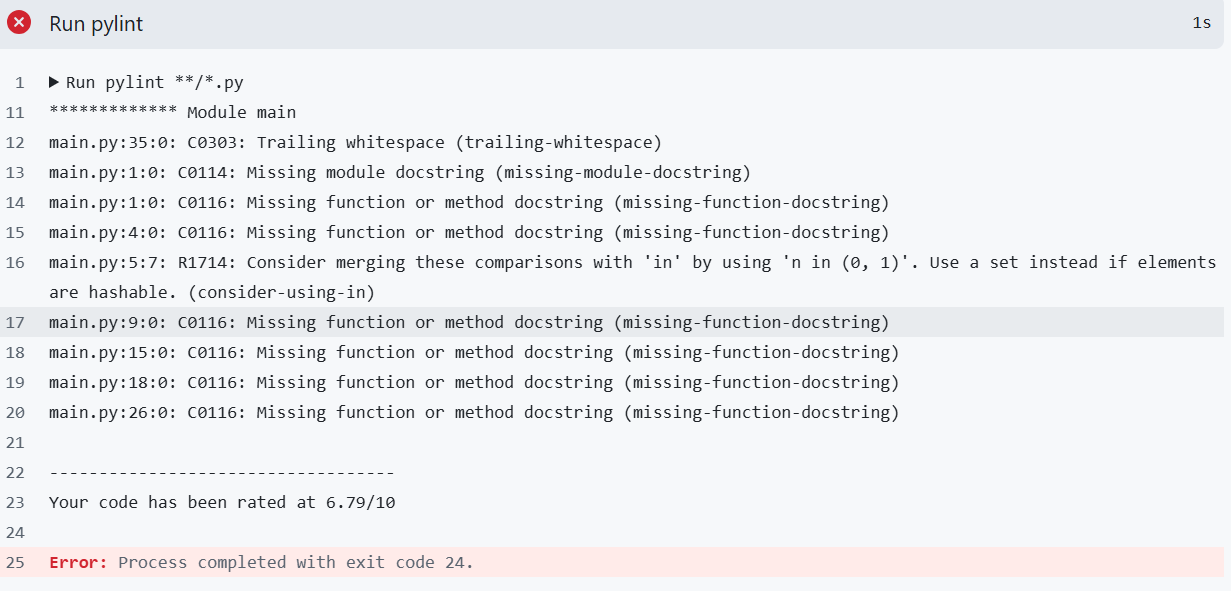
* Pulling changes:

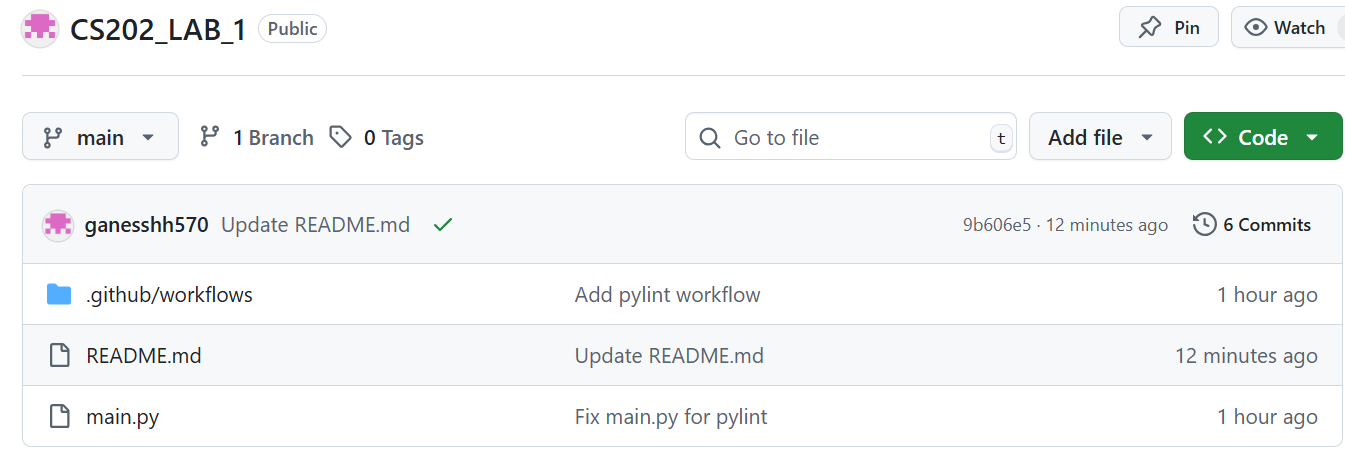
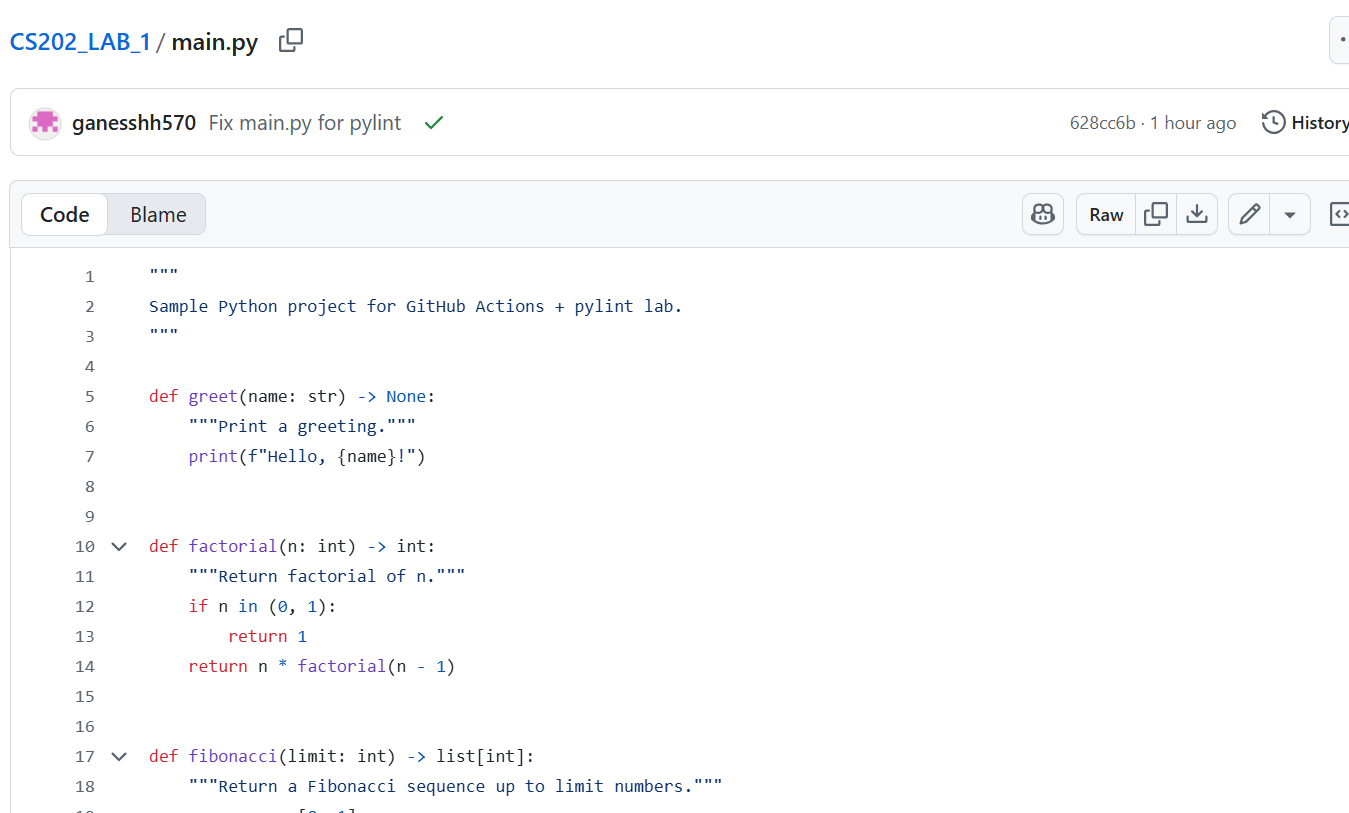


## Setting up pylint



Got a docstring error after adding it error is resolved





# Results and Analysis Outputs

* Successfully cloned a remote repository to the local system, demonstrating the ability to access and work on existing projects.
* Created, switched, and merged multiple branches, showing hands-on experience with parallel development and version management.
* Committed changes with descriptive messages, tracked project history, and resolved merge conflicts, reflecting practical understanding of collaboration and conflict resolution.
* Generated a clear commit history that can be reviewed to understand project progress and modifications over time.

# Observations

* observed that Git effectively tracks every change, making it easy to revert mistakes and maintain project history.
* Using branches allows multiple team members to work simultaneously without overwriting each other’s work, highlighting how collaboration becomes structured and manageable.
* Merge conflicts are inevitable in collaborative work, and resolving them taught us the significance of clear commit messages, regular updates, and communication.

# Key Insights

* I realized that Git is not just for storing code but also for tracking every change, which makes it easier to manage projects over time.
* Working with GitHub taught me how collaboration is handled in real projects, especially when dealing with pulls, pushes, and conflicts.
* Setting up GitHub Actions with pylint showed me the value of automation, as it continuously checks code quality and helps maintain professional standards.

# Discussion and Conclusion Challenges

* During the lab, I explored how Git helps in tracking changes, managing versions, and working collaboratively through GitHub.
* I practiced the full workflow starting from creating a local repo, committing files, linking with GitHub, and pushing updates.
* I also experimented with GitHub Actions, which gave me an understanding of how automation ensures better code quality.

# Reflections

* I learned how to use Git for version control by creating repositories, committing changes, and syncing with GitHub.
* I understood how to handle remote workflows, including pushing, pulling, and resolving merge conflicts.
* I gained hands-on experience with GitHub Actions and pylint, which taught me the importance of automated code quality checks.

# Lessons learned

* Version control's significance in cooperative development.
* the advantages of utilising GitHub Actions and other continuous integration tools.
* Successful branch management strategies to prevent dispute

# Summary

# In this lab, I practiced using Git and GitHub for version control. I created a local repository, added and committed files, and then pushed them to a remote GitHub repository. I also resolved merge conflicts while pulling changes and wrote a Python program with more than 30 lines. Finally, I set up a GitHub Actions workflow with pylint and fixed errors until I got a green tick, which gave me good hands-on experience with Git, GitHub, and automated code checks.

# Summary

# In this lab, I practiced using Git and GitHub for version control. I created a local repository, added and committed files, and then pushed them to a remote GitHub repository. I also resolved merge conflicts while pulling changes and wrote a Python program with more than 30 lines. Finally, I set up a GitHub Actions workflow with pylint and fixed errors until I got a green tick, which gave me good hands-on experience with Git, GitHub, and automated code checks.