# Source Code Management

Name :Adithya K.P

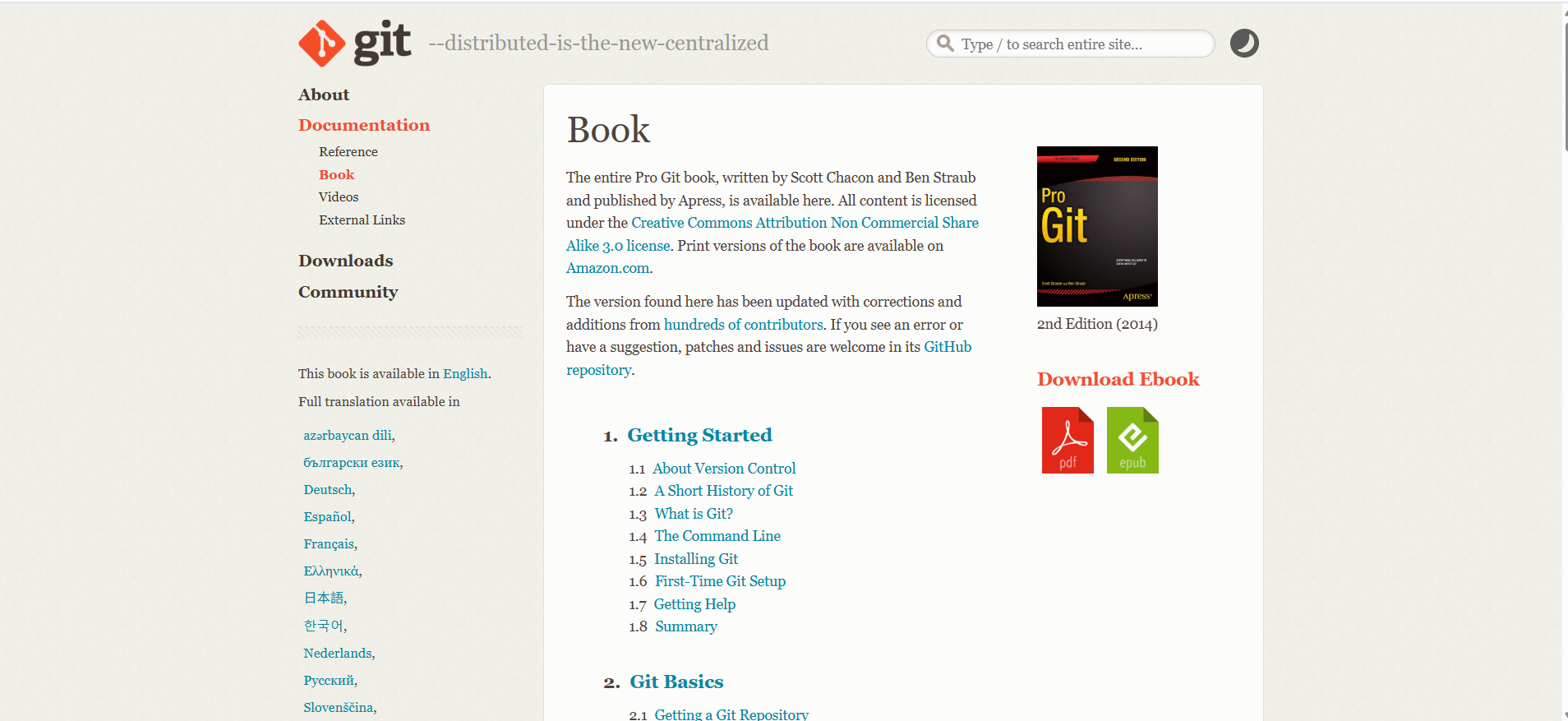
Branch : Btech CSE(AI & ML)

# SEN:A866175124058

# Lab Report 1

# Git Bash Installation

Git Bash is a terminal program for Windows environments that emulates bash command-line experience, allowing you to use Git commands.   
Follow these steps to install Git Bash:  
1. Download the Git installer from the official website (https://git-scm.com/downloads)  
2. Run the installer and follow the installation wizard  
3. During installation, choose the appropriate options based on your preferences  
4. Once installed, you can access Git Bash from the Start menu or by right-clicking in a folder and selecting "Git Bash Here"  
 litre



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# Lab Report 2

Git Confiig

Open Git Bash

• Go to the Start Menu (Windows).

• Type "Git Bash" and click to open it.

Set Your Name

This name will appear in your commits.

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

Set Your Email Address

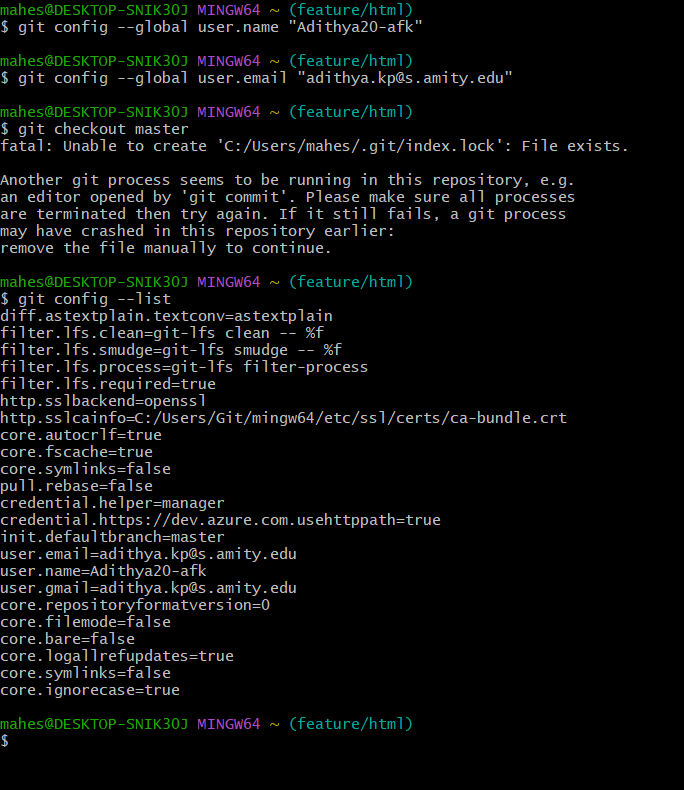
This should match the email used on your GitHub or Git account.

git config --global user.email youremail@example.com

Check Your Configuration

To confirm that your details were saved:

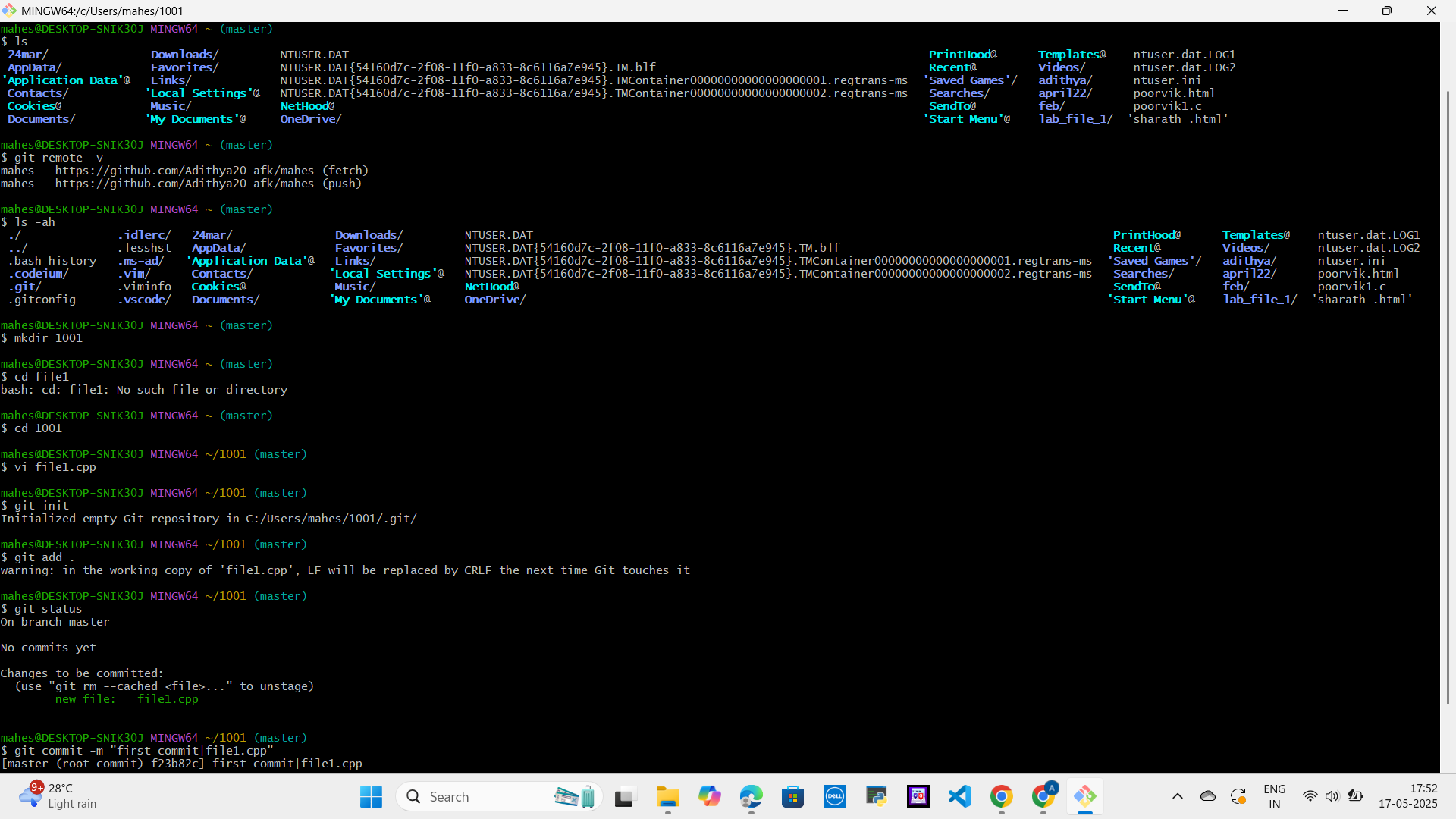
git config --global –list

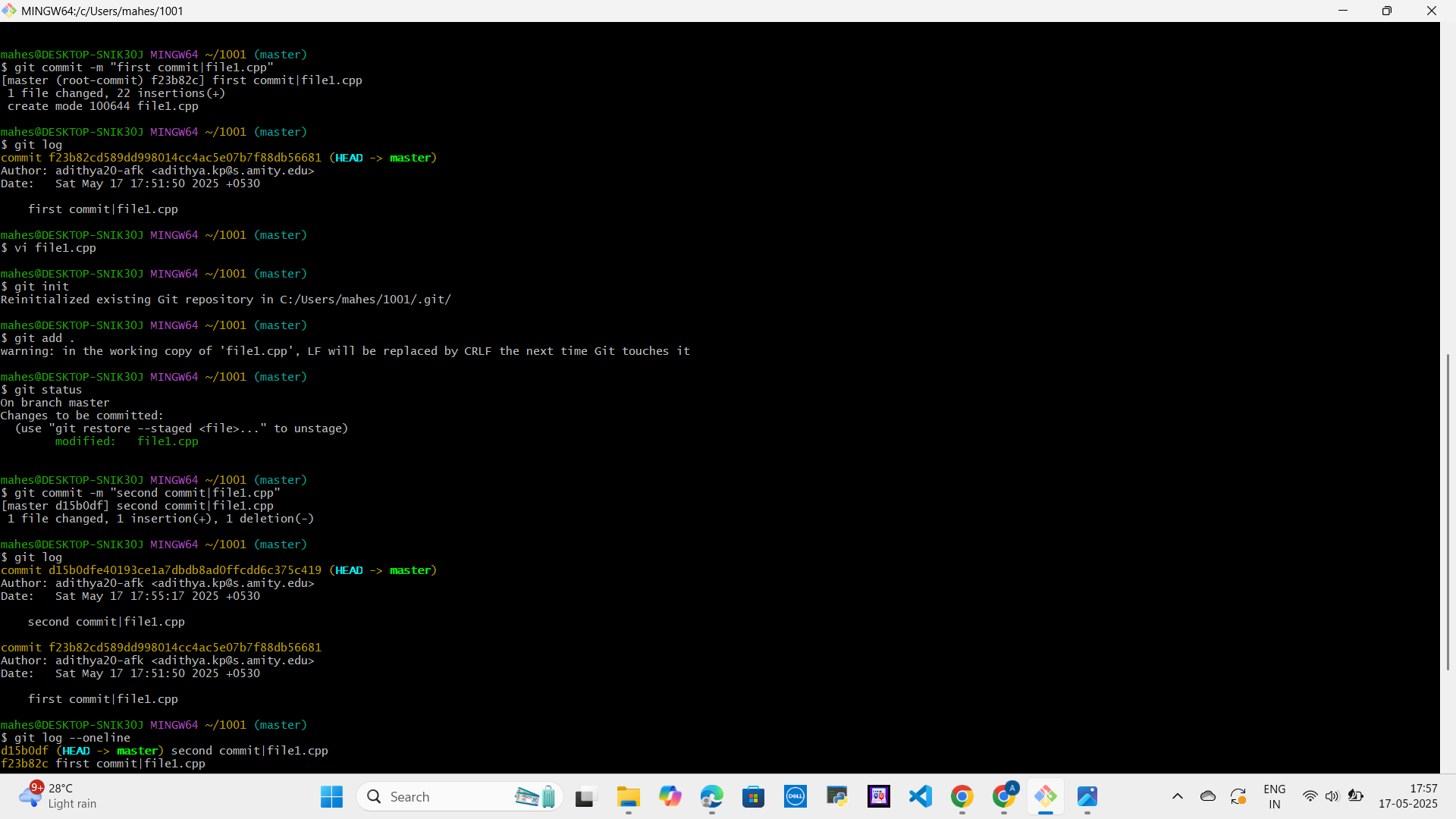


# Lab Report 3

# Git Diff

Git diff shows changes between commits, commit and working tree, etc.  
  
Common uses:  
- git diff: Show unstaged changes  
- git diff --staged: Show staged changes that will be included in the next commit  
- git diff [commit1] [commit2]: Compare two commits  
- git diff [branch1] [branch2]: Compare two branches  
  
This command helps you verify what changes you're about to commit or see what changed between different points in your project history.



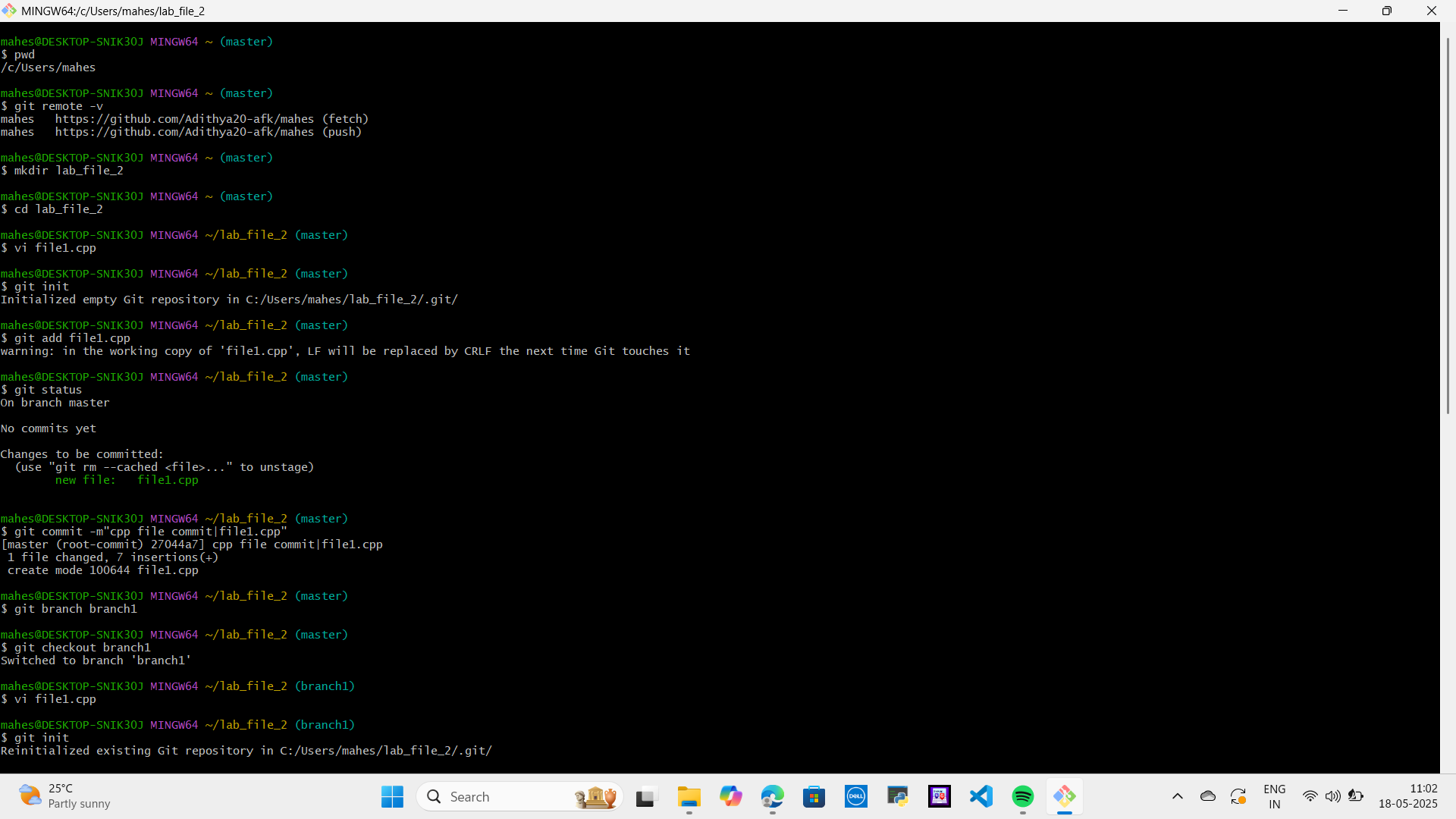


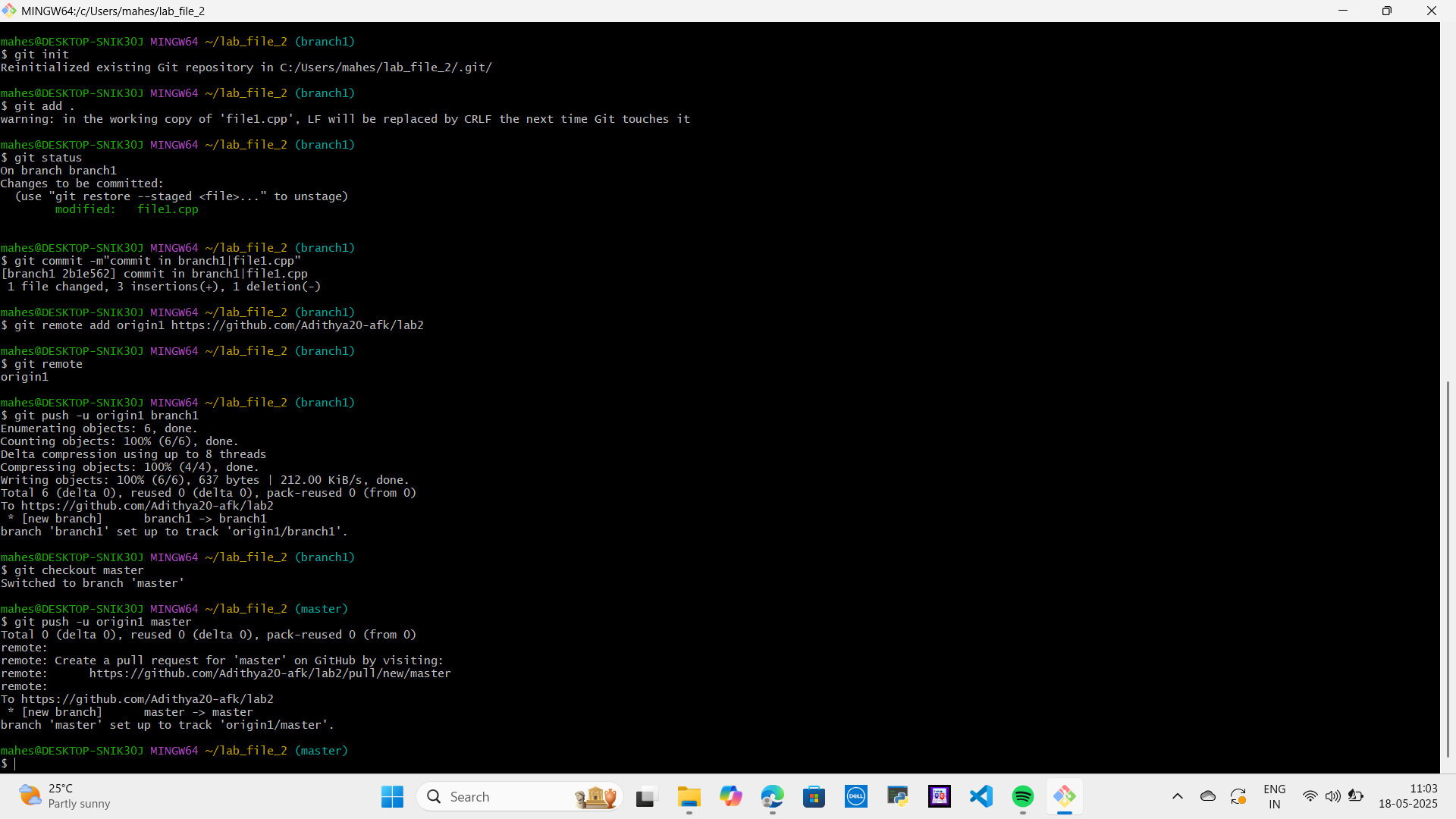


# Lab Report 4

# Git Commit

Git commit records changes to the repository by creating a snapshot of the staged changes.  
  
Basic workflow:  
1. Make changes to your files  
2. Stage changes using git add [filename] or git add .  
3. Commit changes using git commit -m "Commit message"  
  
Best practices:  
- Write clear, concise commit messages  
- Commit related changes together  
- Commit often with smaller, logical changesets  
- Avoid committing incomplete work

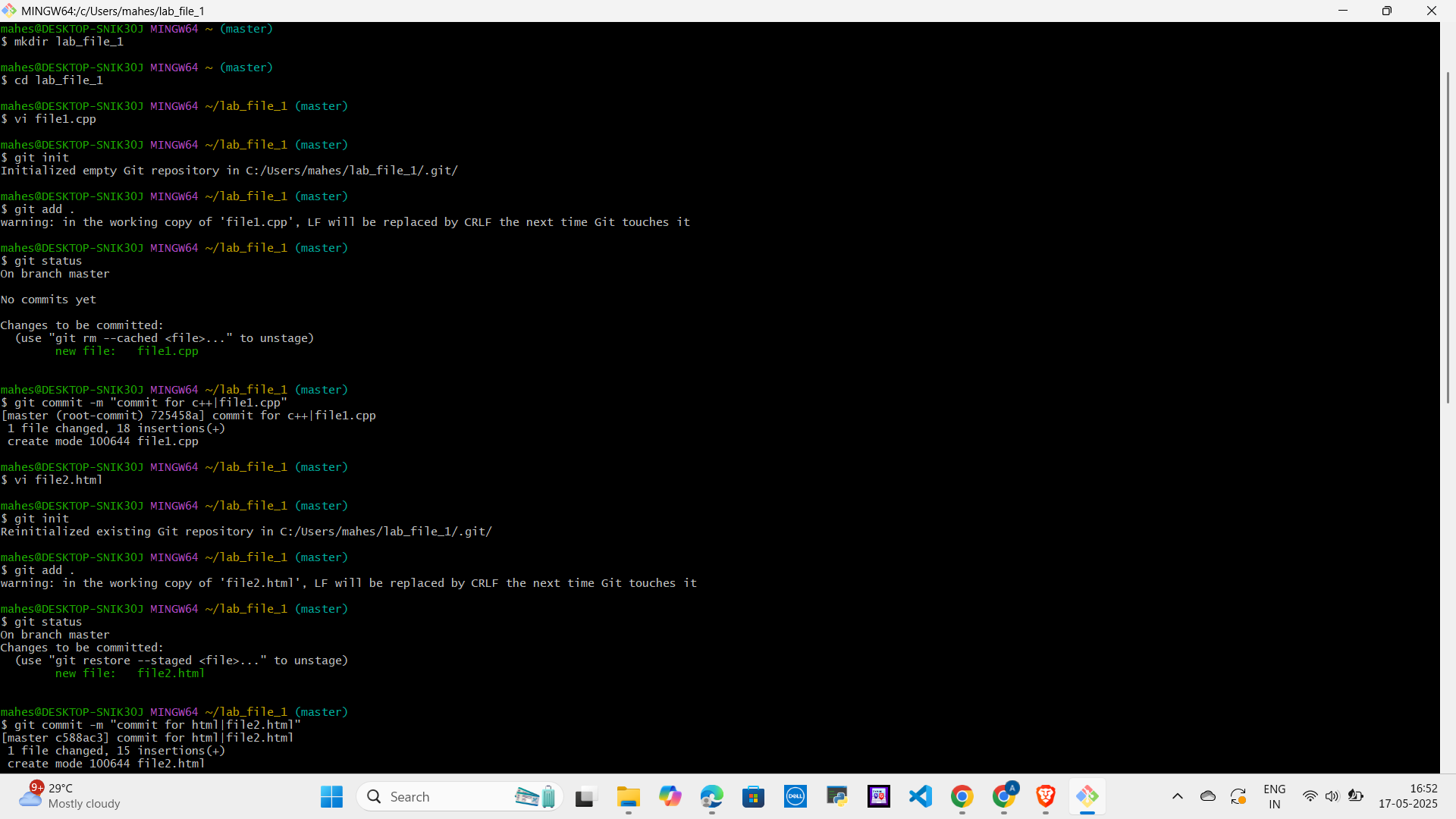


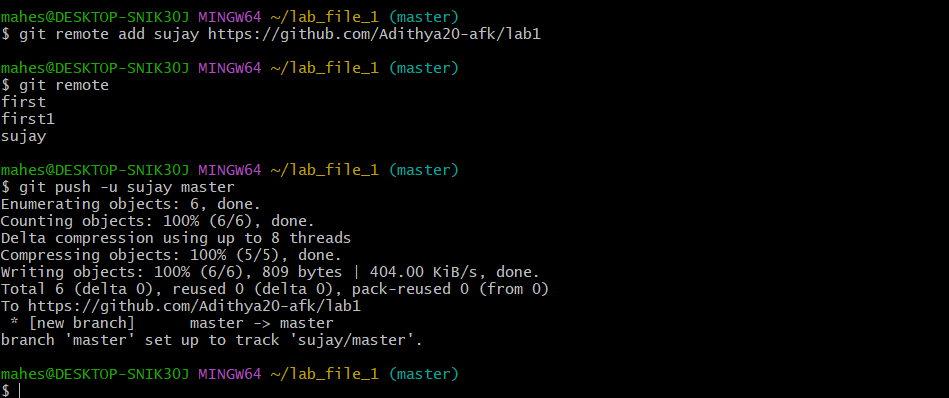


# Lab Report 5

# Git Push

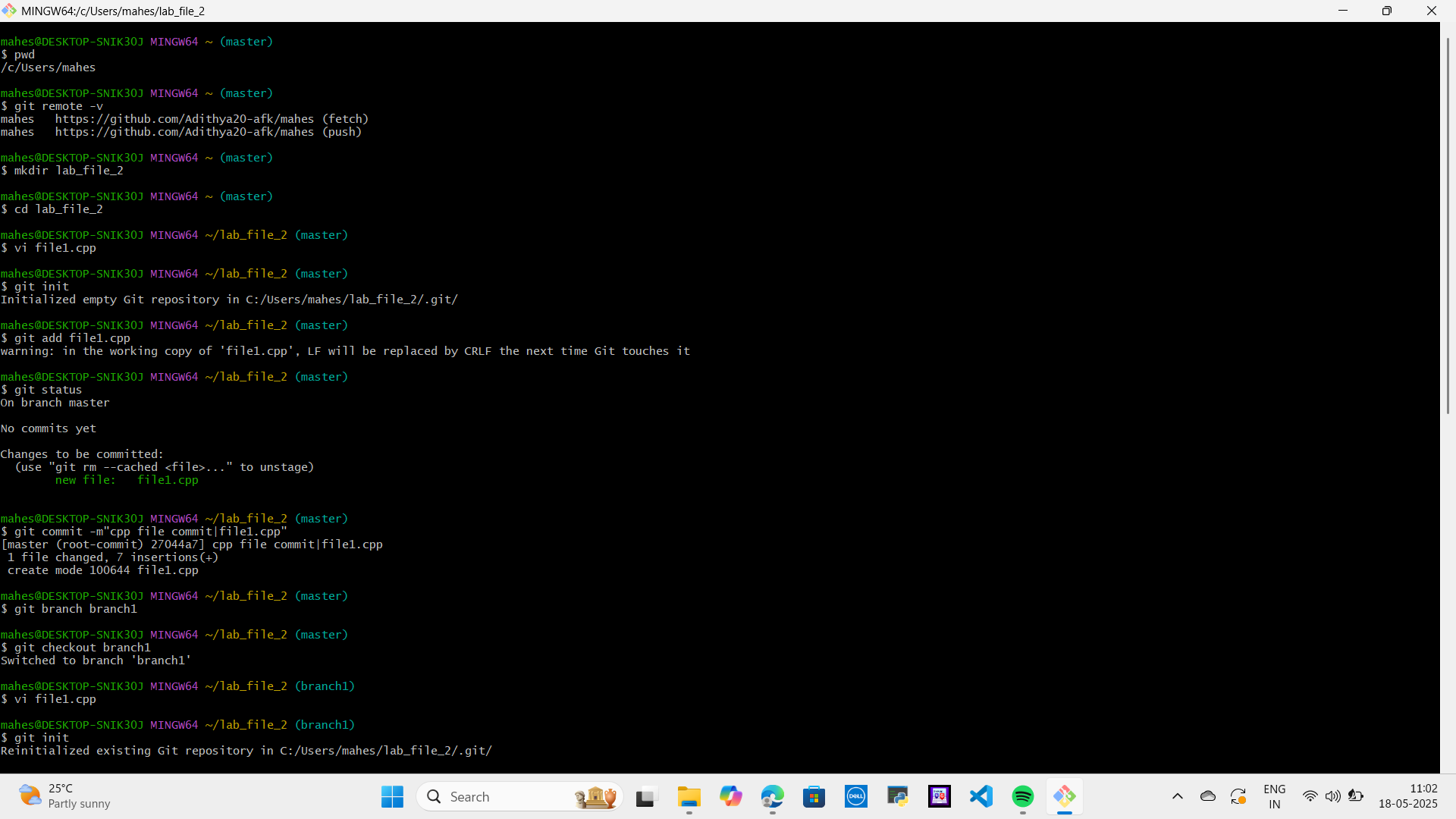
Git push updates the remote repository with your local commits.  
  
Syntax: git push [remote] [branch]  
  
Example: git push origin main  
  
This command:  
- Uploads your commits to the remote repository  
- Updates the remote tracking branches  
- Requires proper permissions on the remote repository  
- May require resolving conflicts if the remote has changes your local doesn't  
  
Common options:  
- git push -u origin [branch]: Set upstream tracking reference  
- git push --force: Force push (use with caution!)  
- git push --all: Push all branches

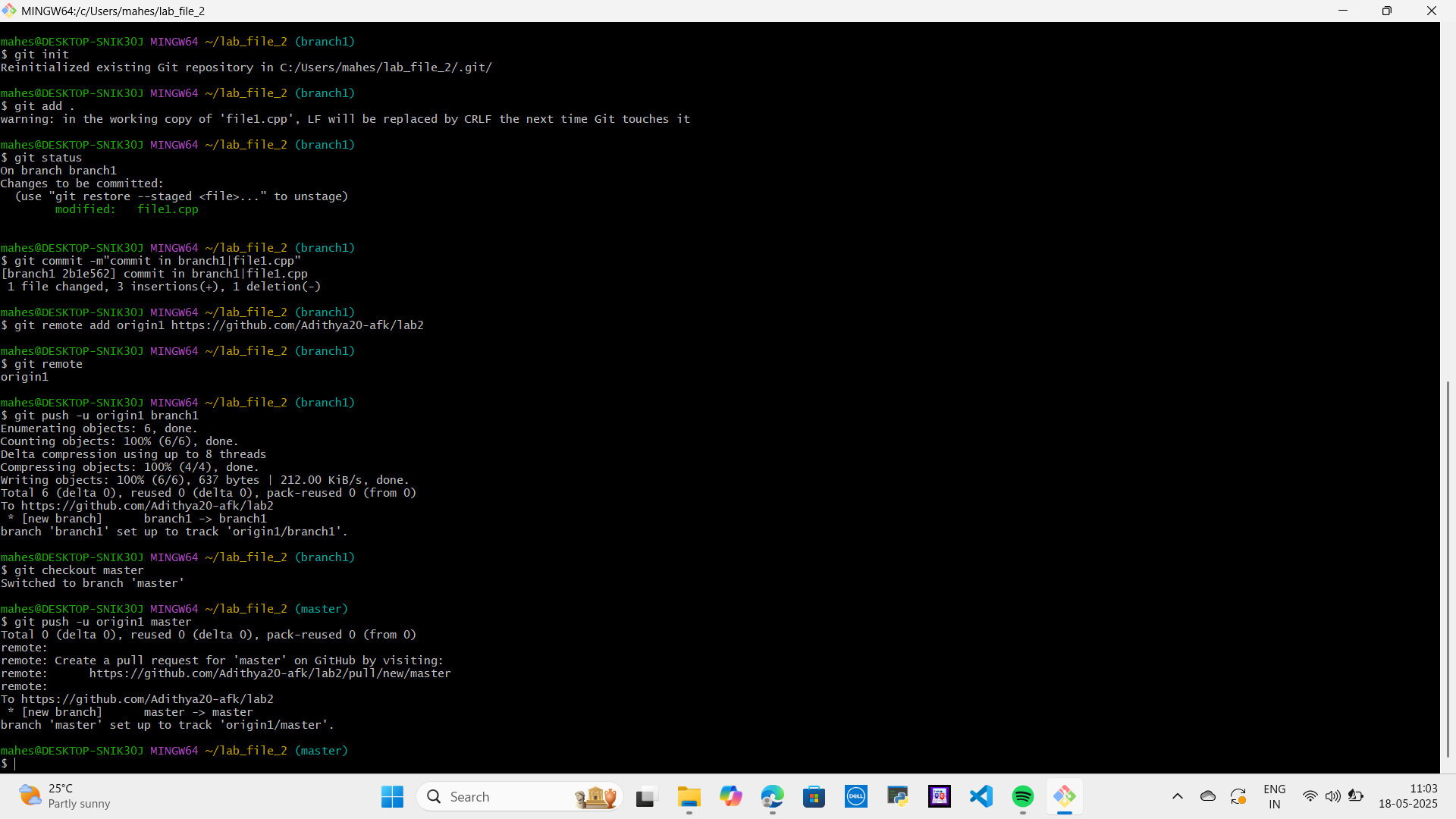




# Lab Report 6

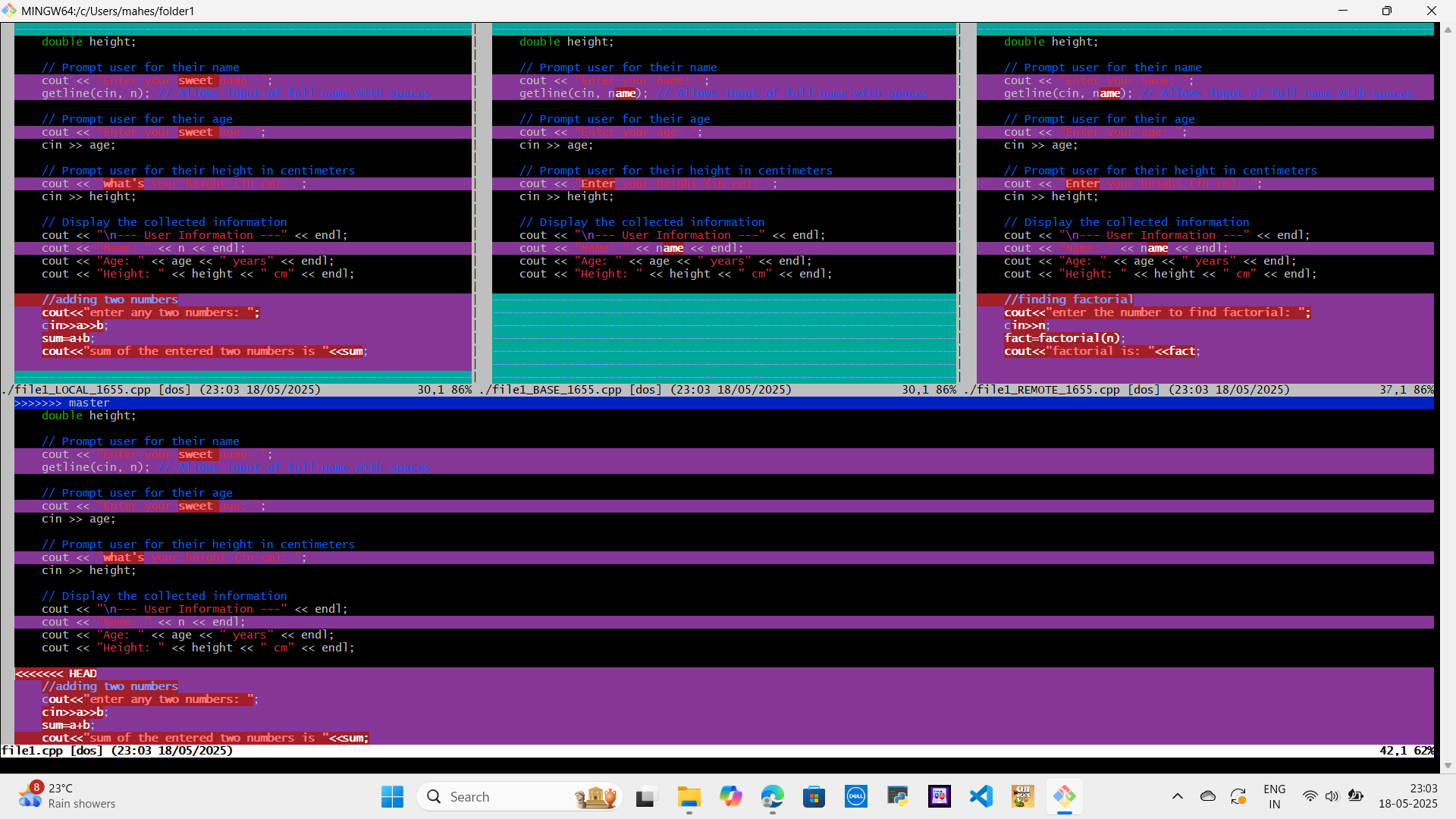
# Git Branch

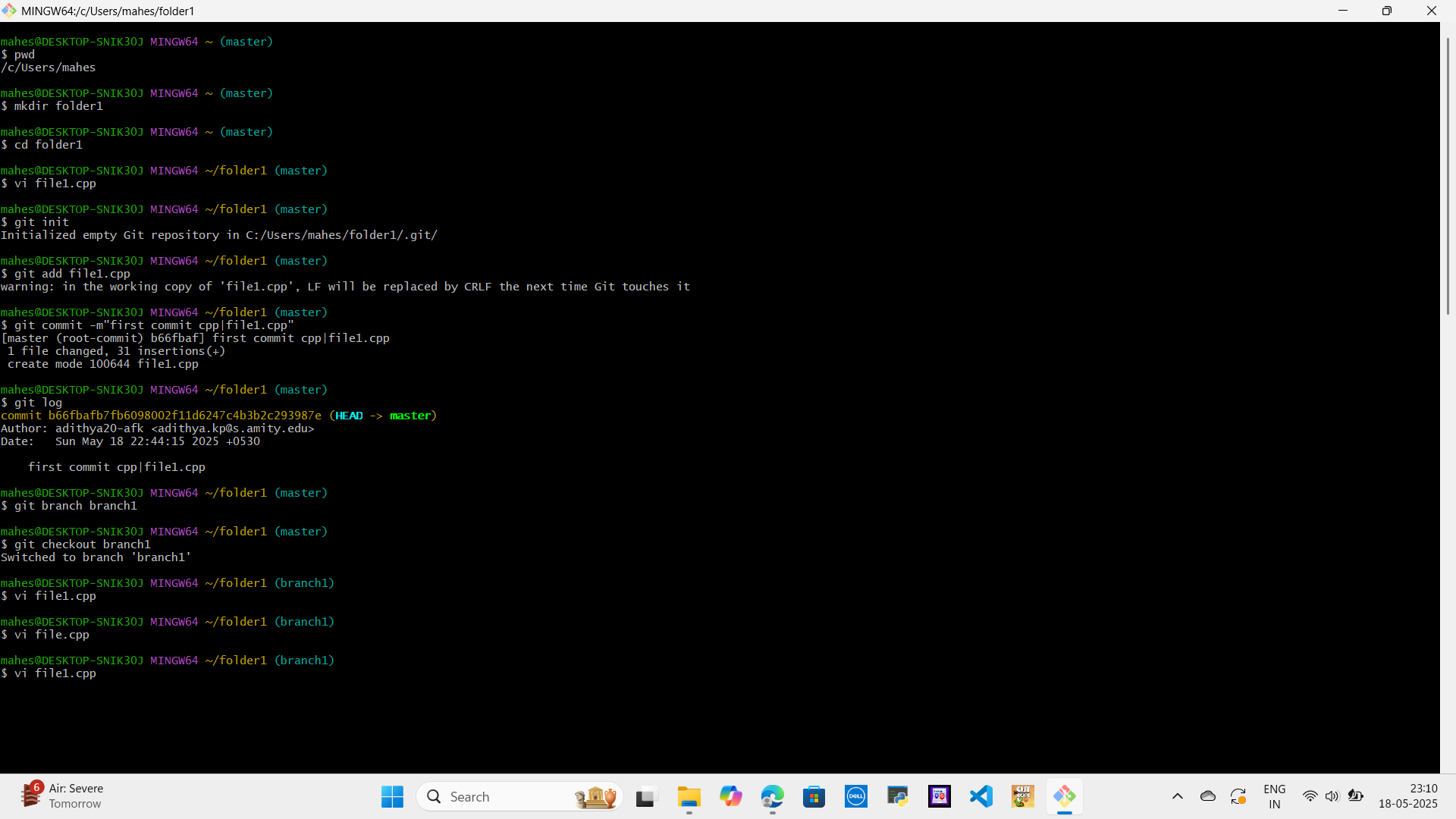
Git branches allow you to develop features, fix bugs, or experiment safely without affecting the main codebase.  
  
Common commands:  
- git branch: List all local branches  
- git branch [branch-name]: Create a new branch  
- git branch -d [branch-name]: Delete a branch  
- git branch -a: List all branches (local and remote)  
- git checkout [branch-name]: Switch to a branch  
- git checkout -b [branch-name]: Create and switch to a new branch  
  
Branches are powerful for team collaboration and feature development isolation.  


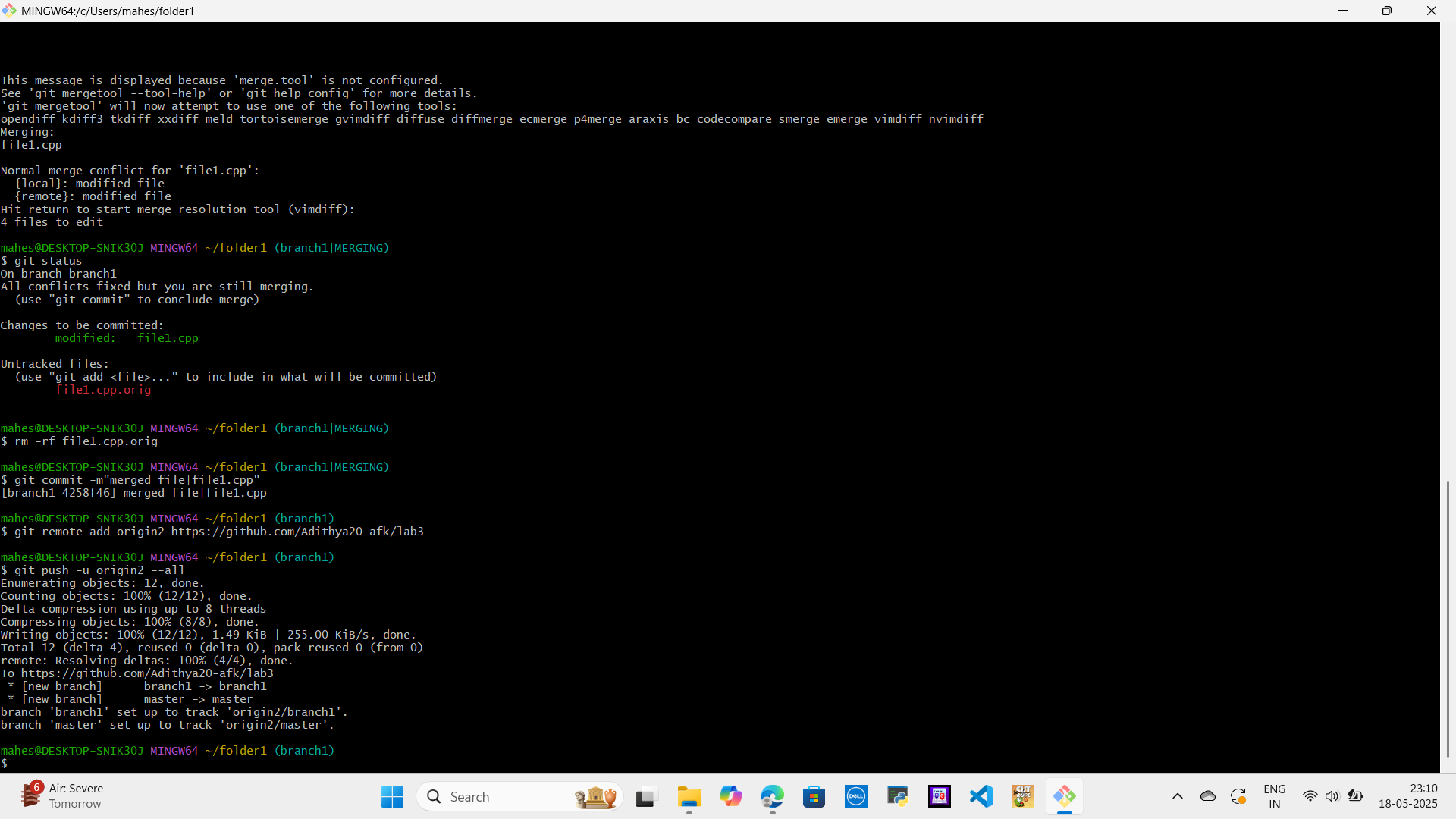


# Lab Report 7

# Git Merge

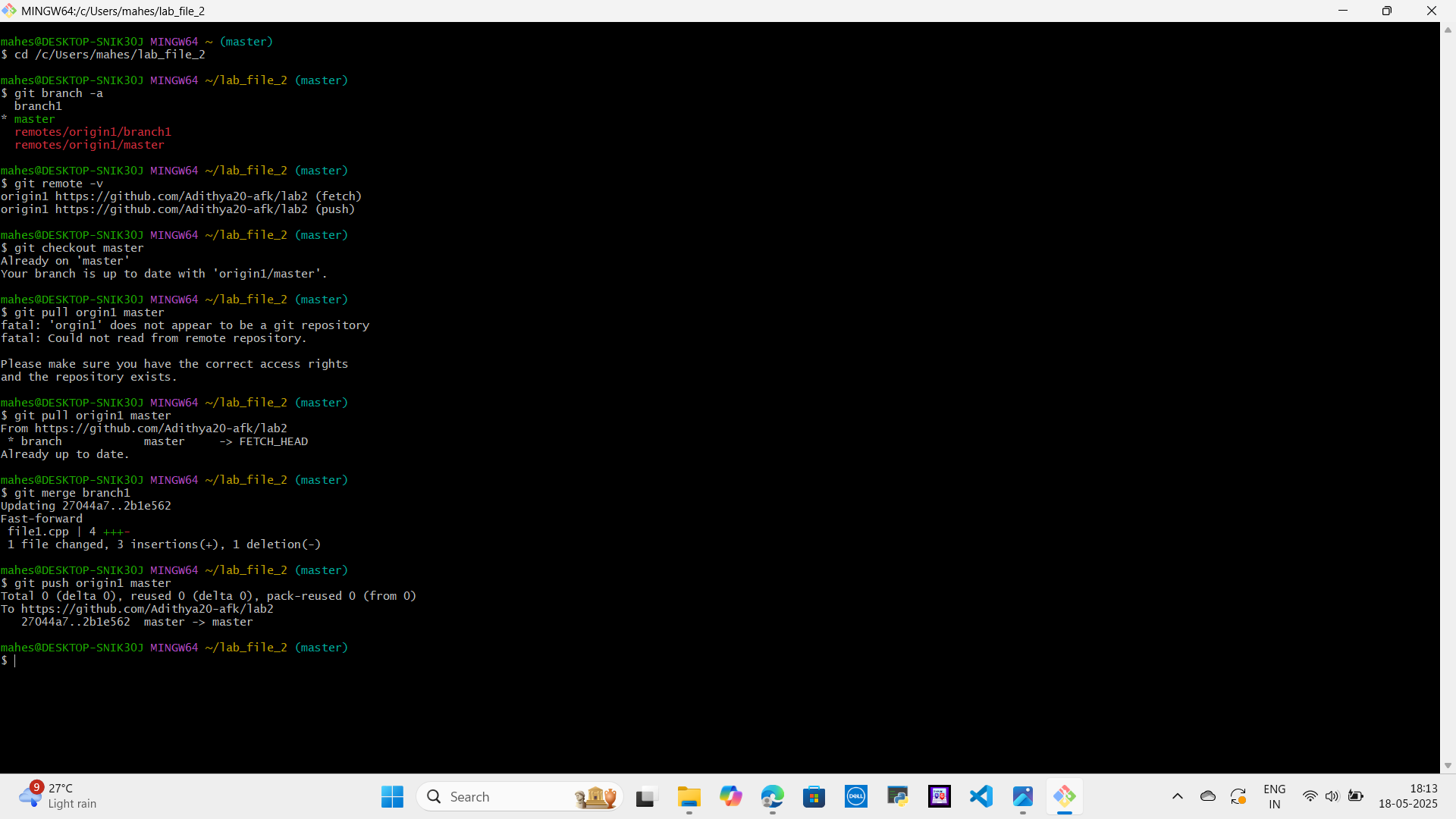
Git merge combines changes from different branches.  
  
Syntax: git merge [branch-name]  
  
Process:  
1. Checkout the branch you want to merge into (e.g., git checkout main)  
2. Run git merge [branch-to-merge]  
3. Resolve any conflicts if they occur  
4. Commit the merge if there were conflicts  
  
Merge types:  
- Fast-forward: When the target branch has no unique commits  
- Three-way merge: When both branches have diverged, creating a new merge commit  
- Conflicts: When the same lines of code have been changed in both branches  






# Lab Report 8

Merge without conflict

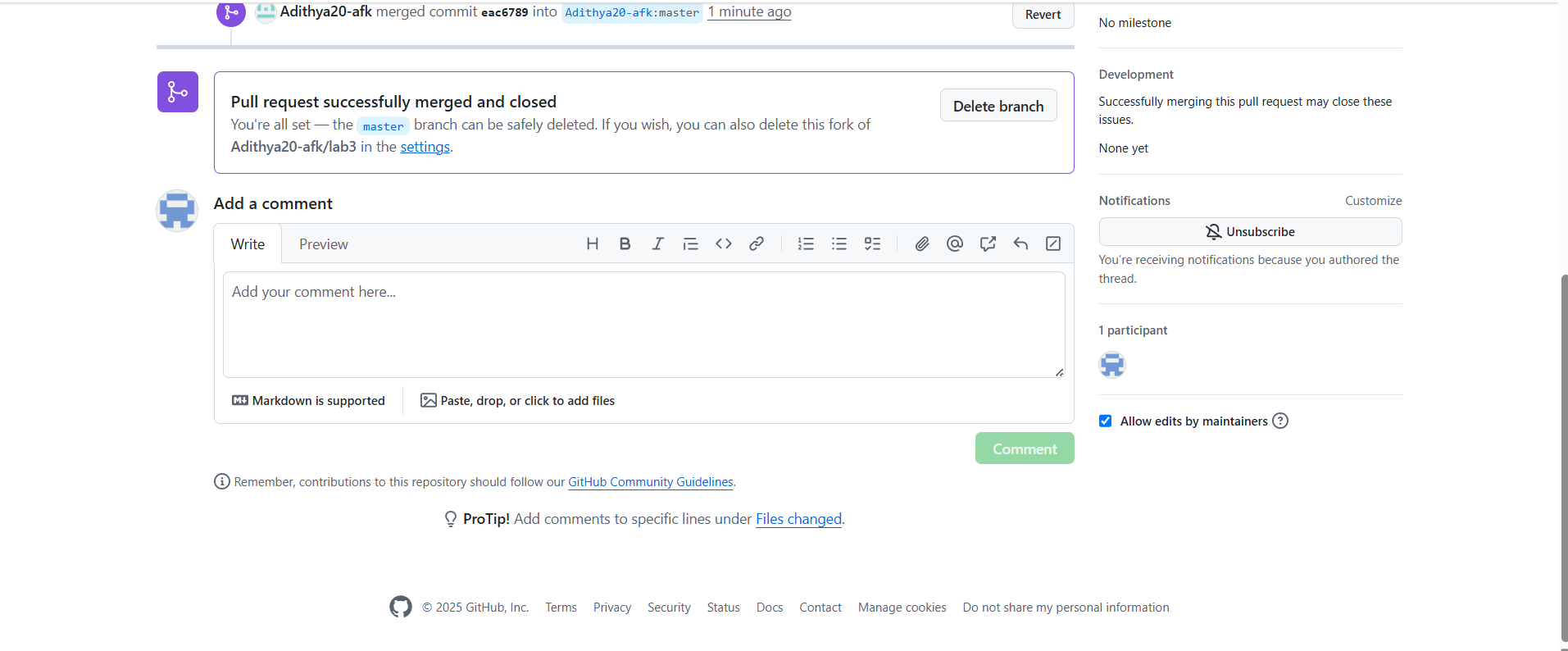


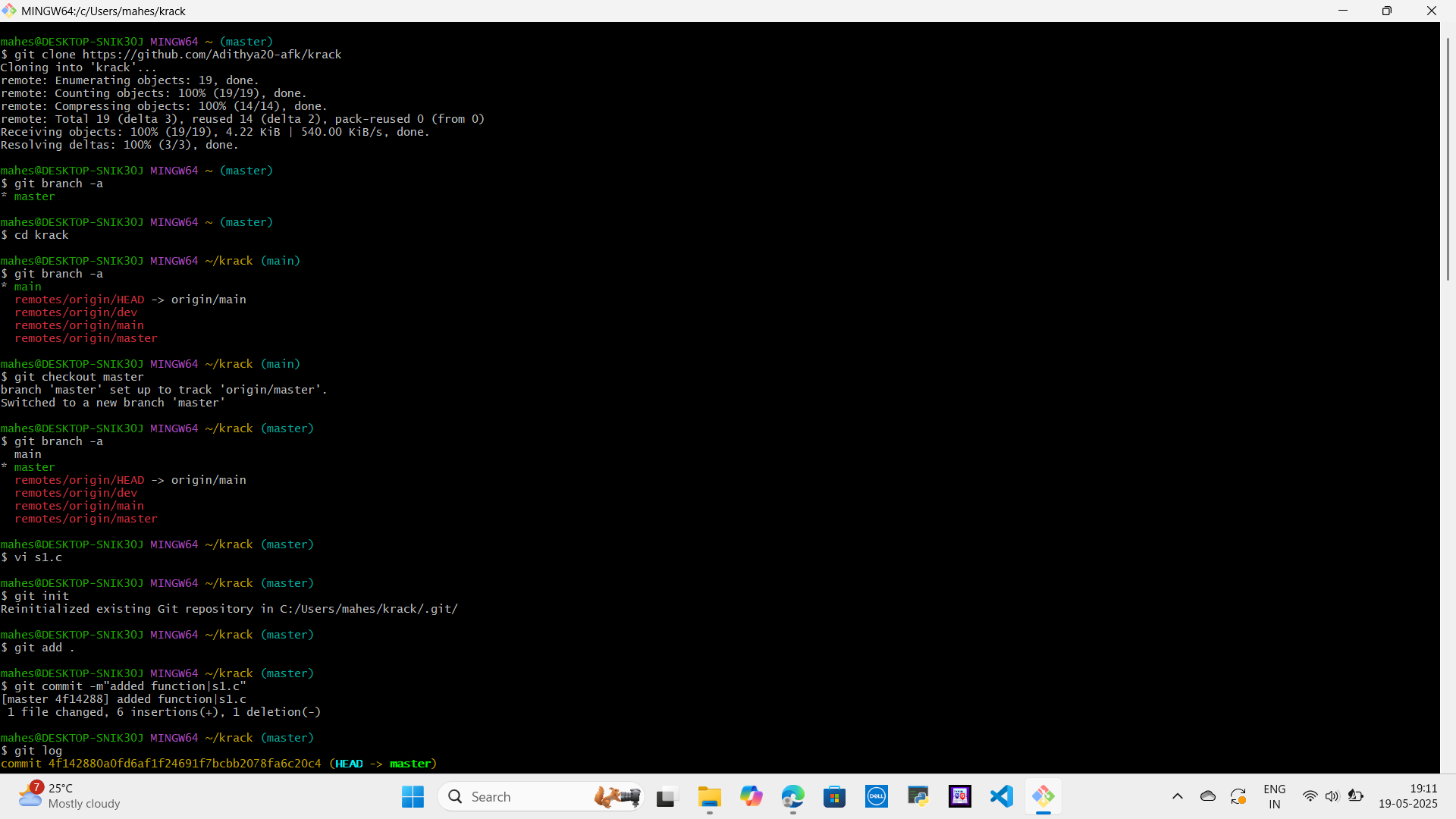
# Lab Report 9

# Git Clone

Git clone creates a copy of an existing repository into a new directory on your local machine.  
  
Syntax: git clone [repository URL]  
  
Example: git clone https://github.com/username/repository.git  
  
This command:  
- Creates a directory with the repository name  
- Initializes a .git directory inside it  
- Pulls all repository data  
- Creates remote-tracking branches  
- Checks out the initial branch (usually 'main' or 'master')

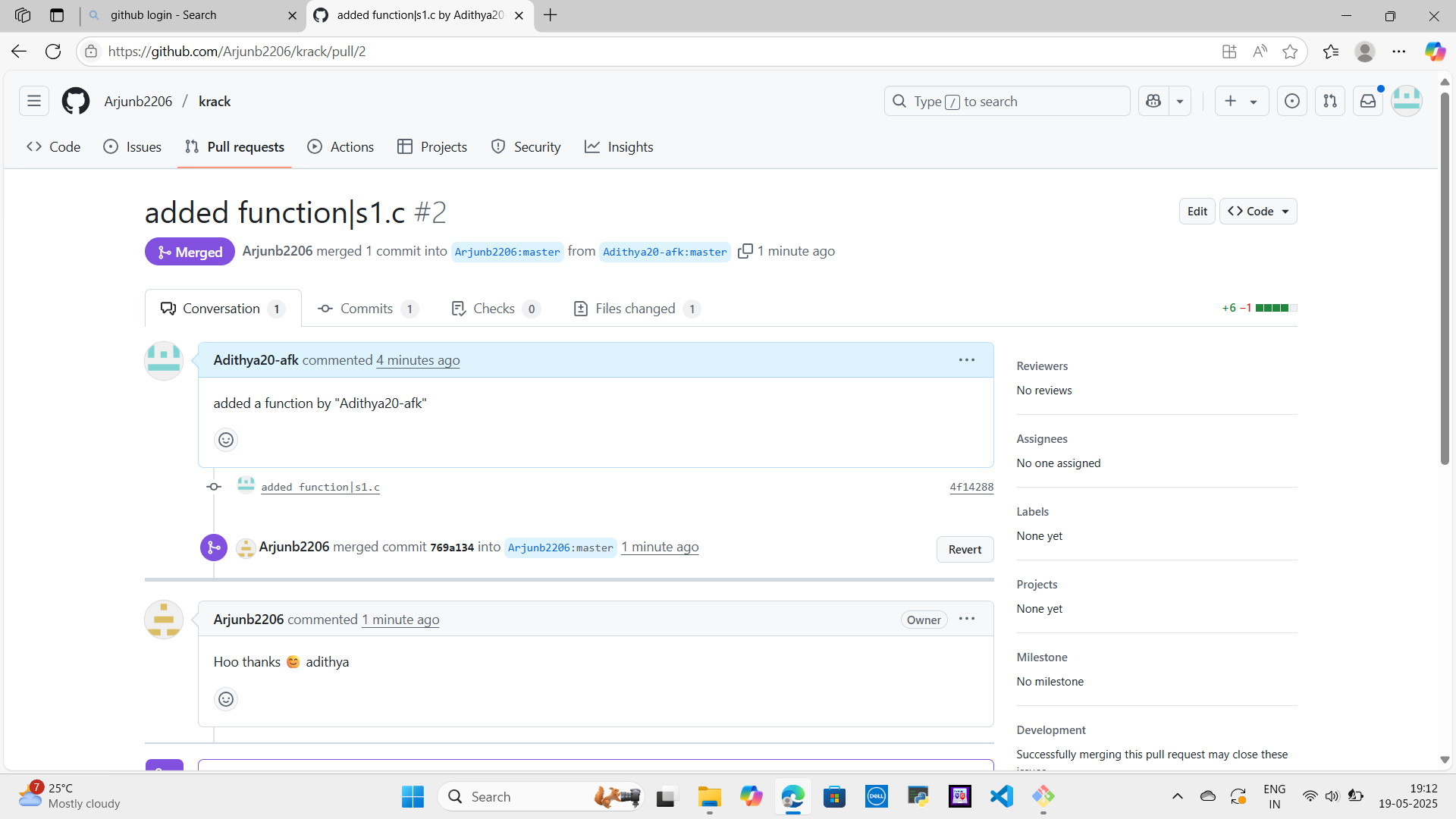
A screenshot of a computer

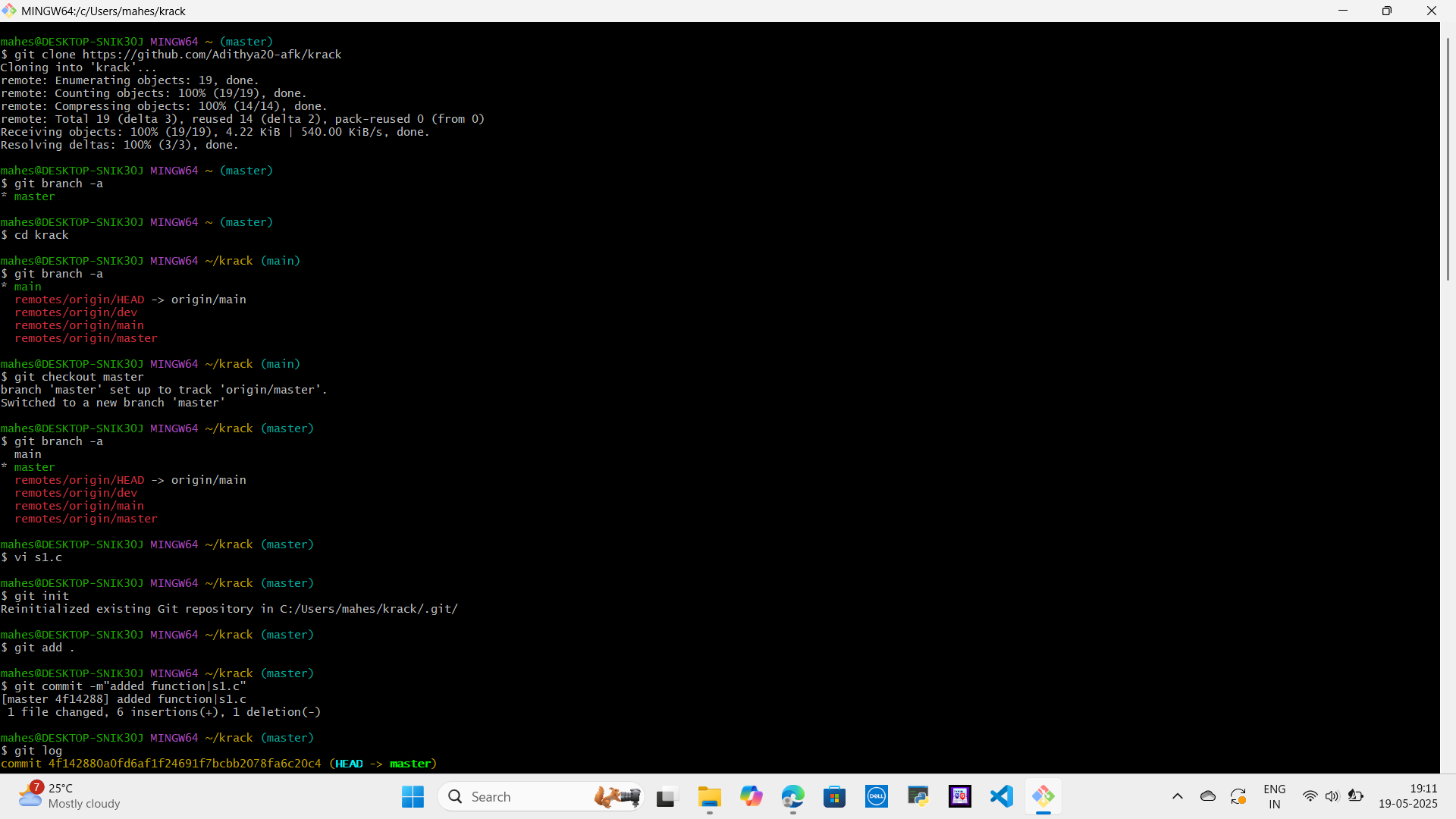
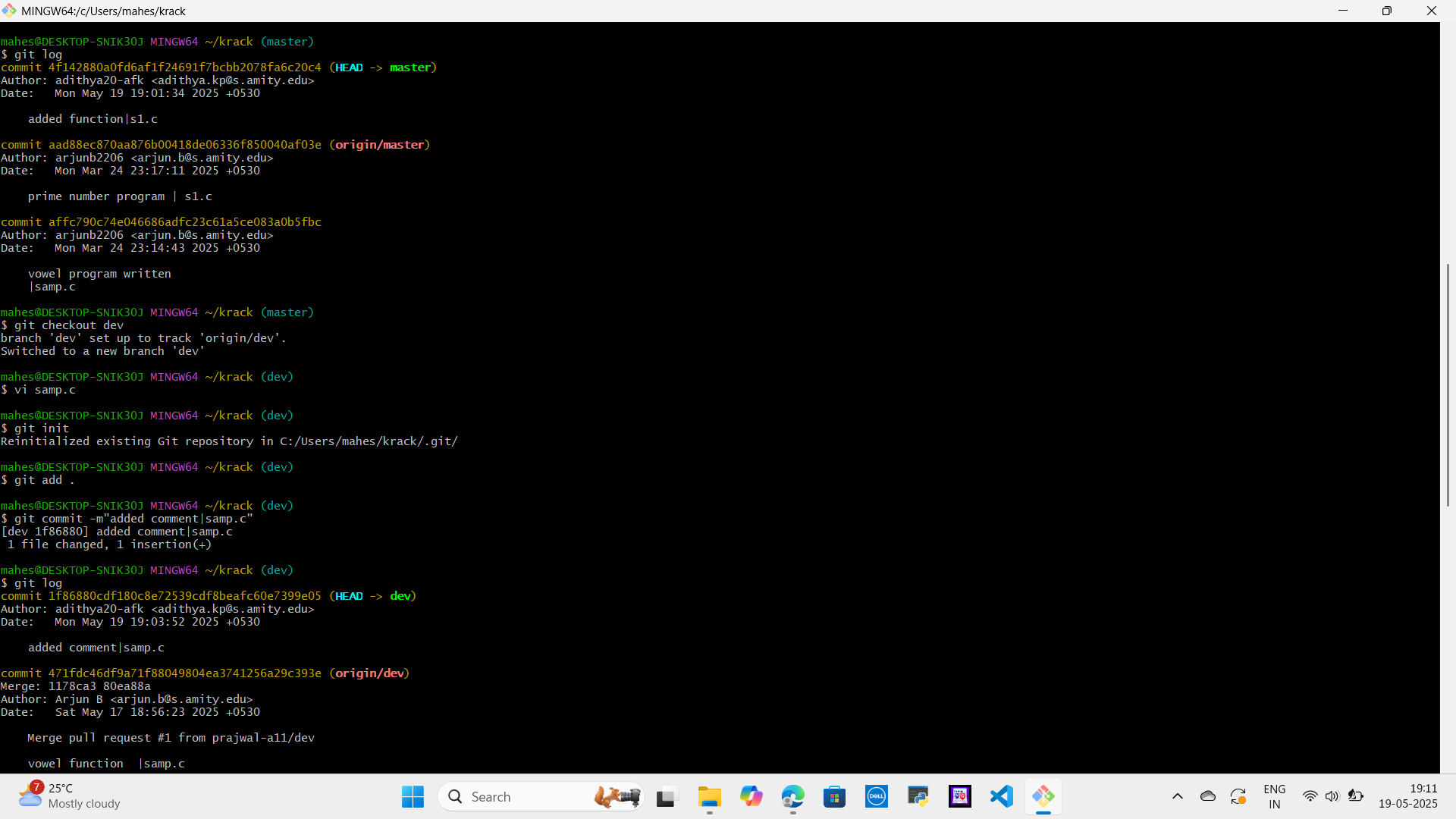
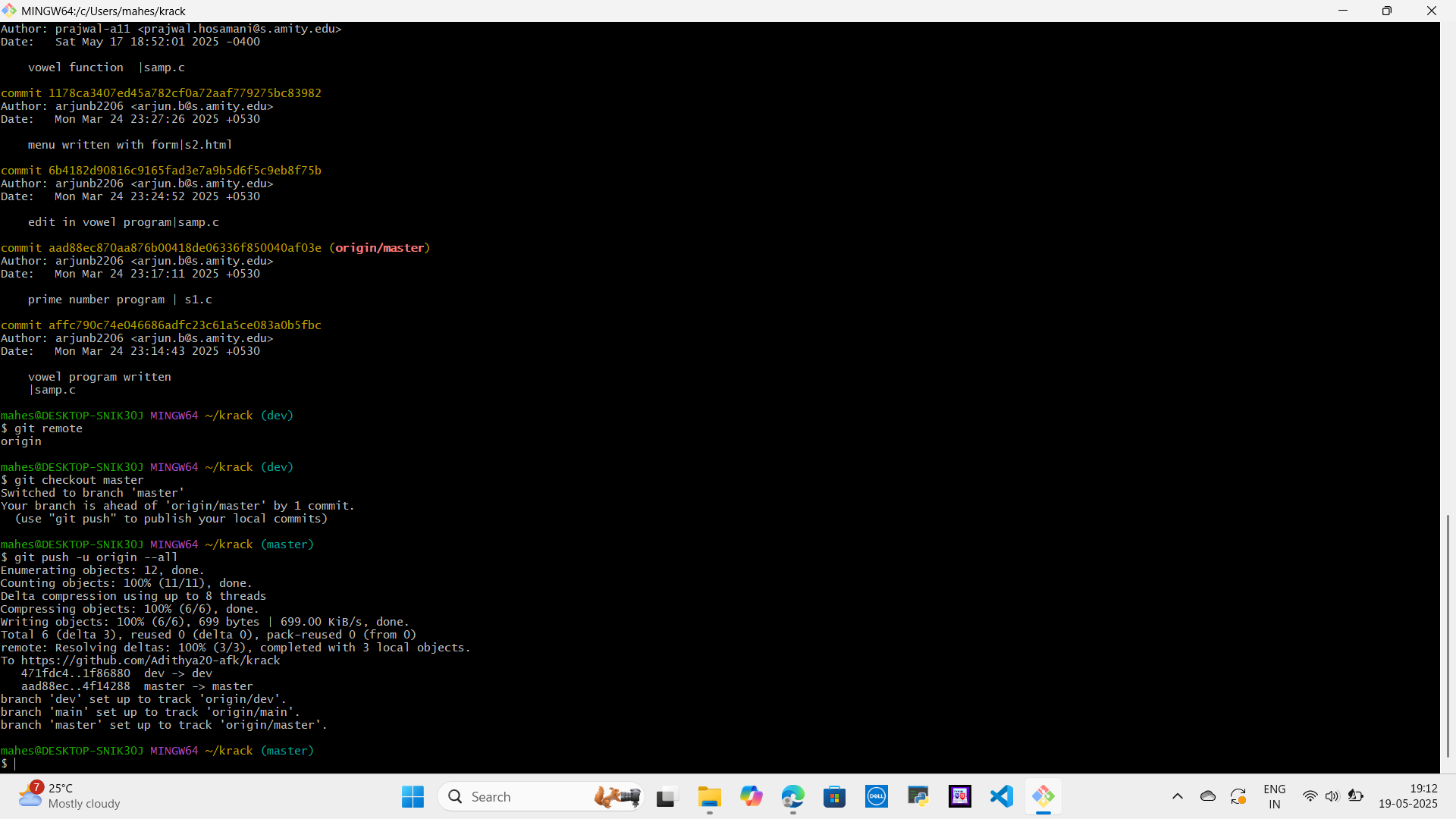
AI-generated content may be incorrect.



# Lab Report 10

# Fork

Forking is a GitHub/GitLab feature that creates a personal copy of someone else's repository under your account.  
  
Key points:  
- Forks allow you to freely experiment without affecting the original project  
- To fork a repository, click the "Fork" button on the repository page  
- Changes made in your fork don't affect the original repository  
- You can submit Pull/Merge Requests to propose changes to the original repository  
- Forks maintain a connection to the original repository, allowing you to sync changes  
  
Forking is essential for contributing to open-source projects or building upon existing code.  




# Lab Report 11

# Git Ignore

Git ignore allows you to specify files and directories that Git should ignore.  
  
Create a .gitignore file in your repository root with patterns of files to ignore:  
  
Common examples:  
- \*.log: Ignore all log files  
- node\_modules/: Ignore the entire node\_modules directory  
- .env: Ignore environment variable files  
- build/: Ignore build outputs  
  
Benefits:  
- Prevents sensitive information from being committed  
- Excludes unnecessary files like build artifacts or dependencies  
- Keeps the repository clean and focused on source code  
- Improves performance by not tracking large or numerous files

