**WEEK – 8 : HANDS-ON EXERCISE**

**GIT**

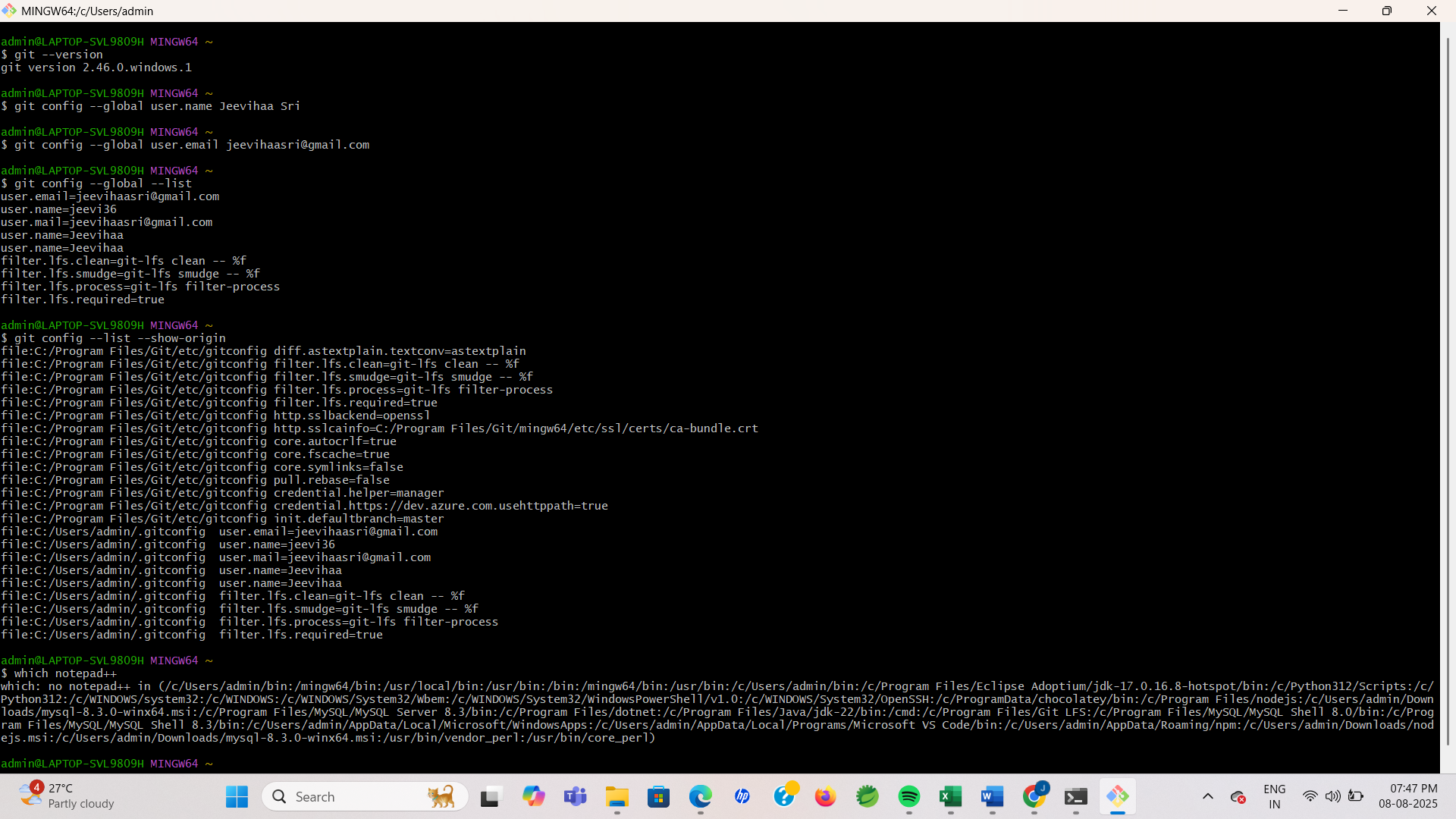
**Exercise 1: Git Configuration and Verification in Git Bash**

**Scenario:**

You need to configure your Git username and email in Git Bash so that all commits are correctly attributed to you in the project history and remote repository.

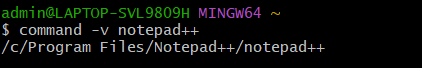
**Step 1: Setup Git configuration**

* Verify Git is installed
* Set your user name and email (global)
* Verify configuration



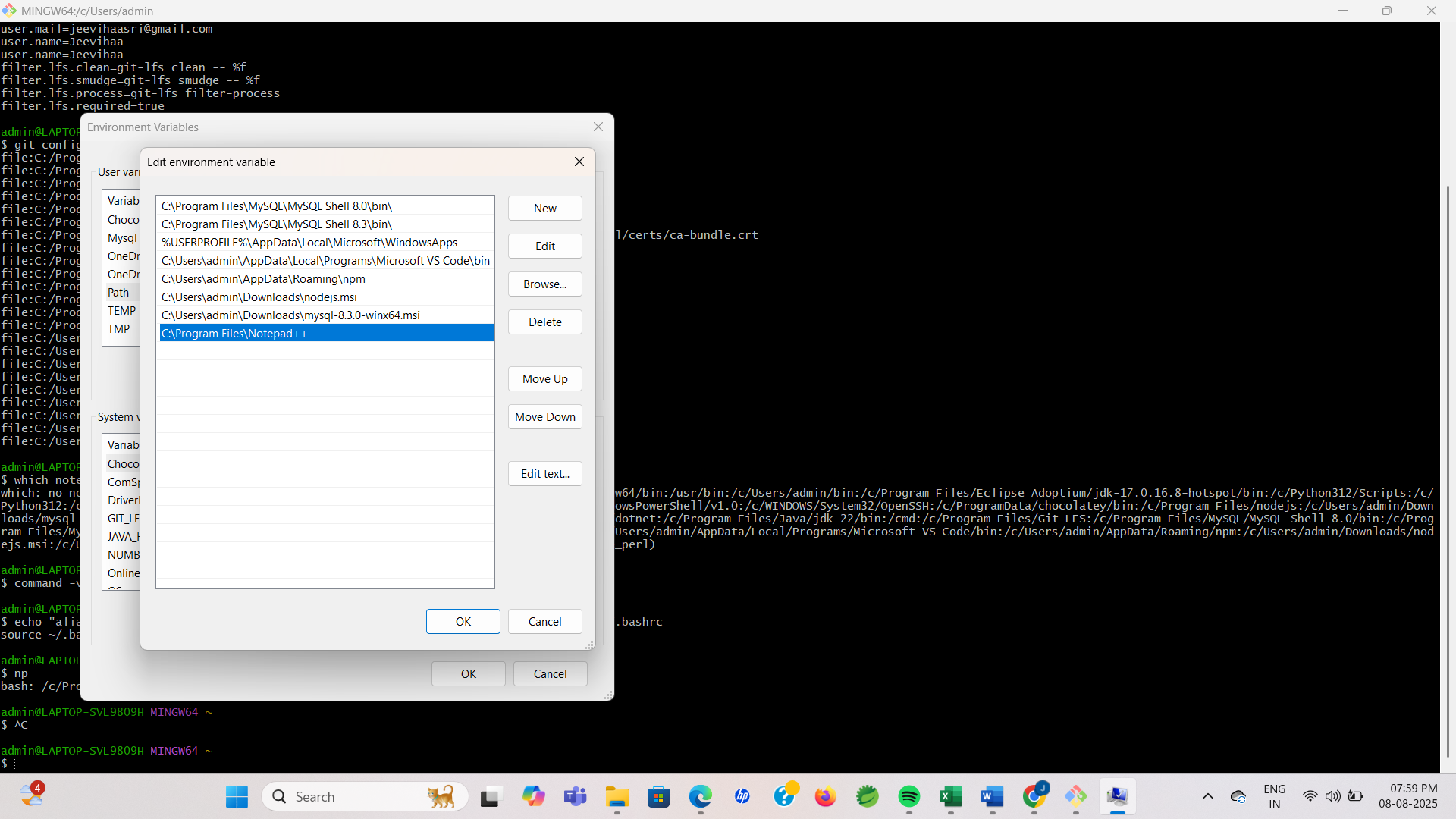
**Step 2: Integrate Notepad++ with Git**

1. Check whether notepad++ is available in Git Bash

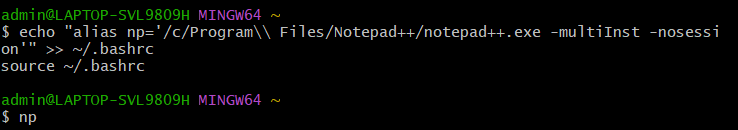


1. If not found, add Notepad++ to your Windows PATH (Permanent):

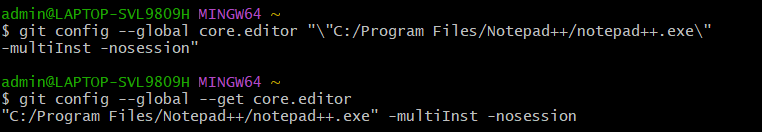
* Open **Control Panel → System → Advanced system settings → Advanced → Environment Variables**
* Under User variables, edit Path and **Add** the folder containing notepad++.exe



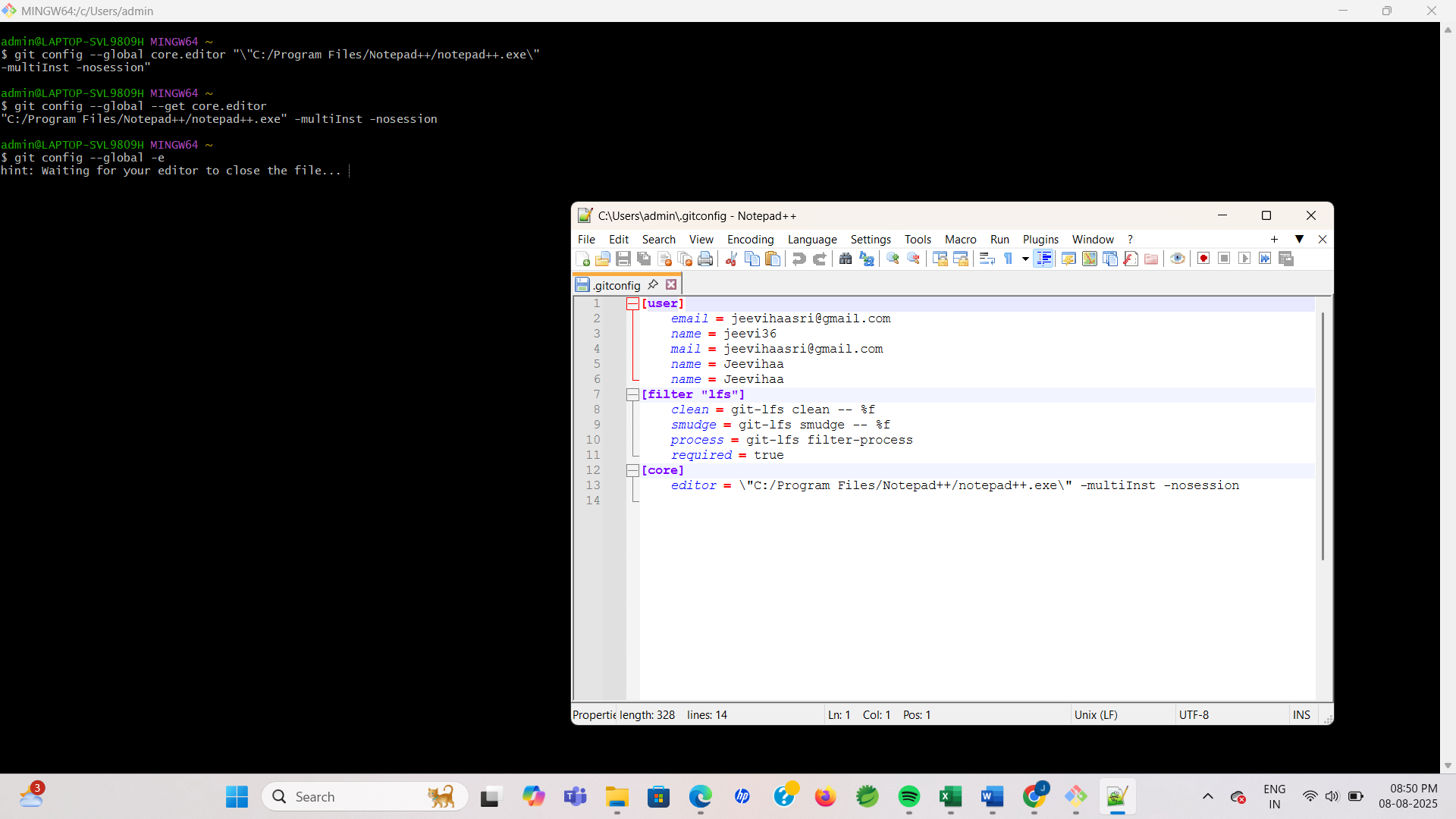
1. **Add a short alias for Notepad++ in Git Bash**



1. **Configure Git to use Notepad++ as the core editor**

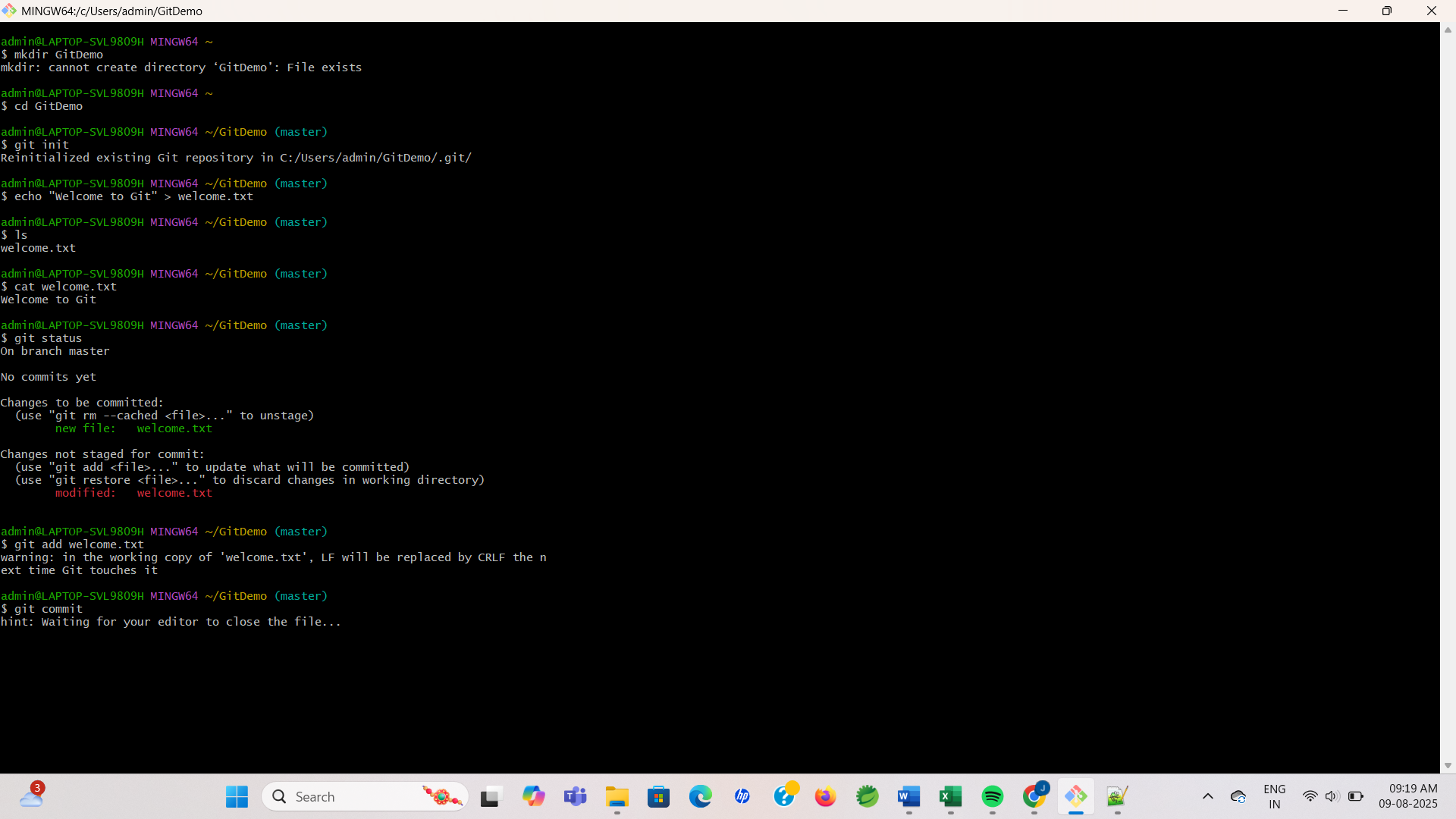
****

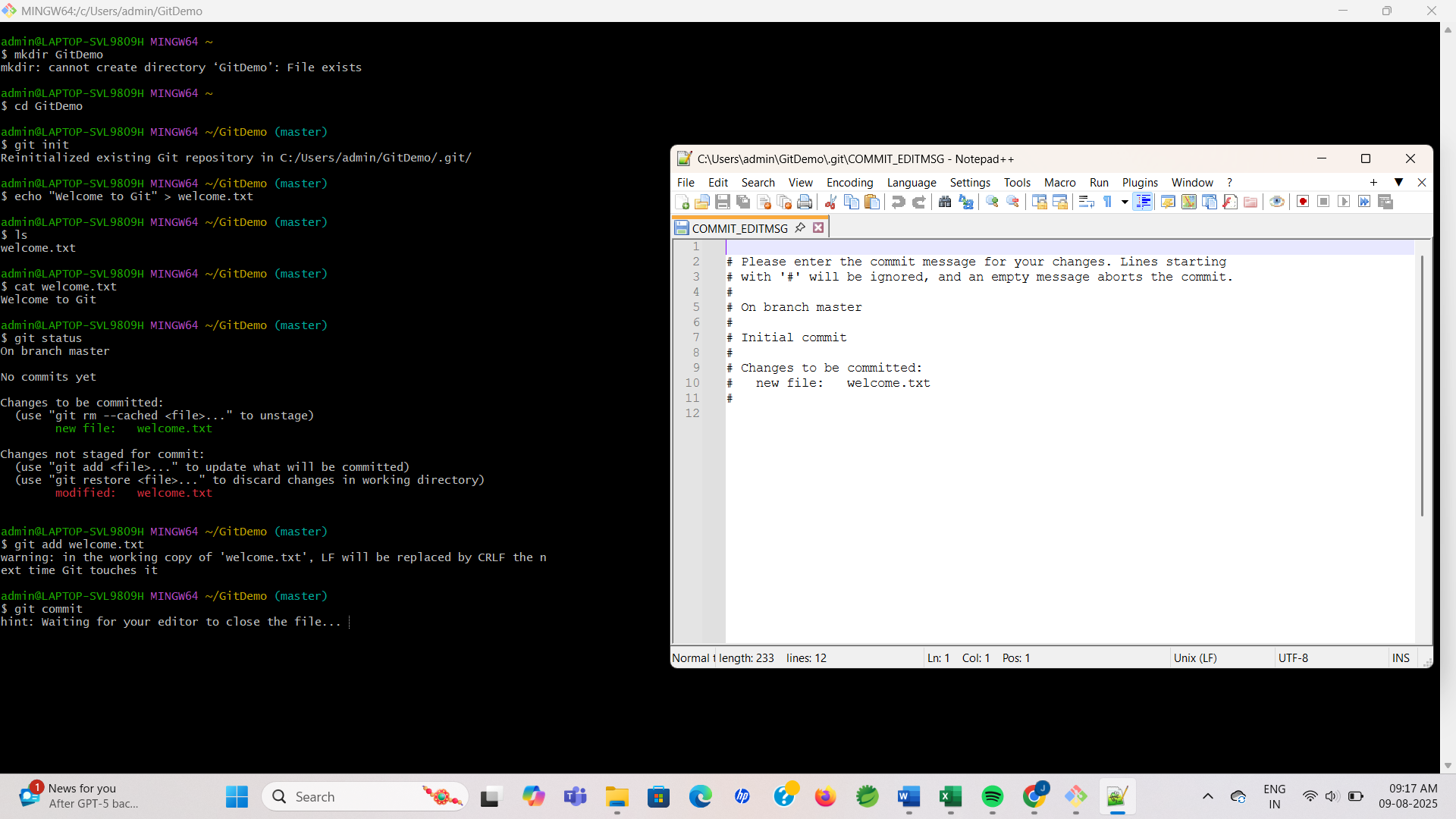
1. **Open the global Git config in your editor**



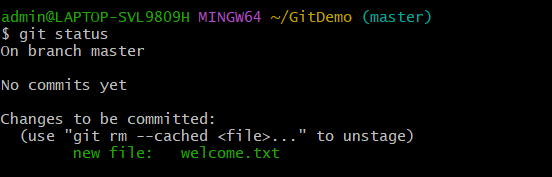
**Step 3: Create and Add Files to a Repository**

1. Create a new folder and open it in Git Bash
2. Initialize the Git repository, Create a file and add content
3. Check the file and Check status
4. Add the file to staging and Commit the changes





1. Check status again

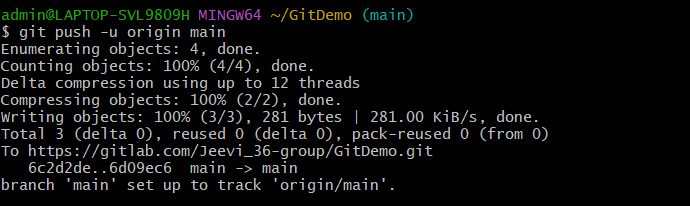


**Step 4: Connect to GitLab**

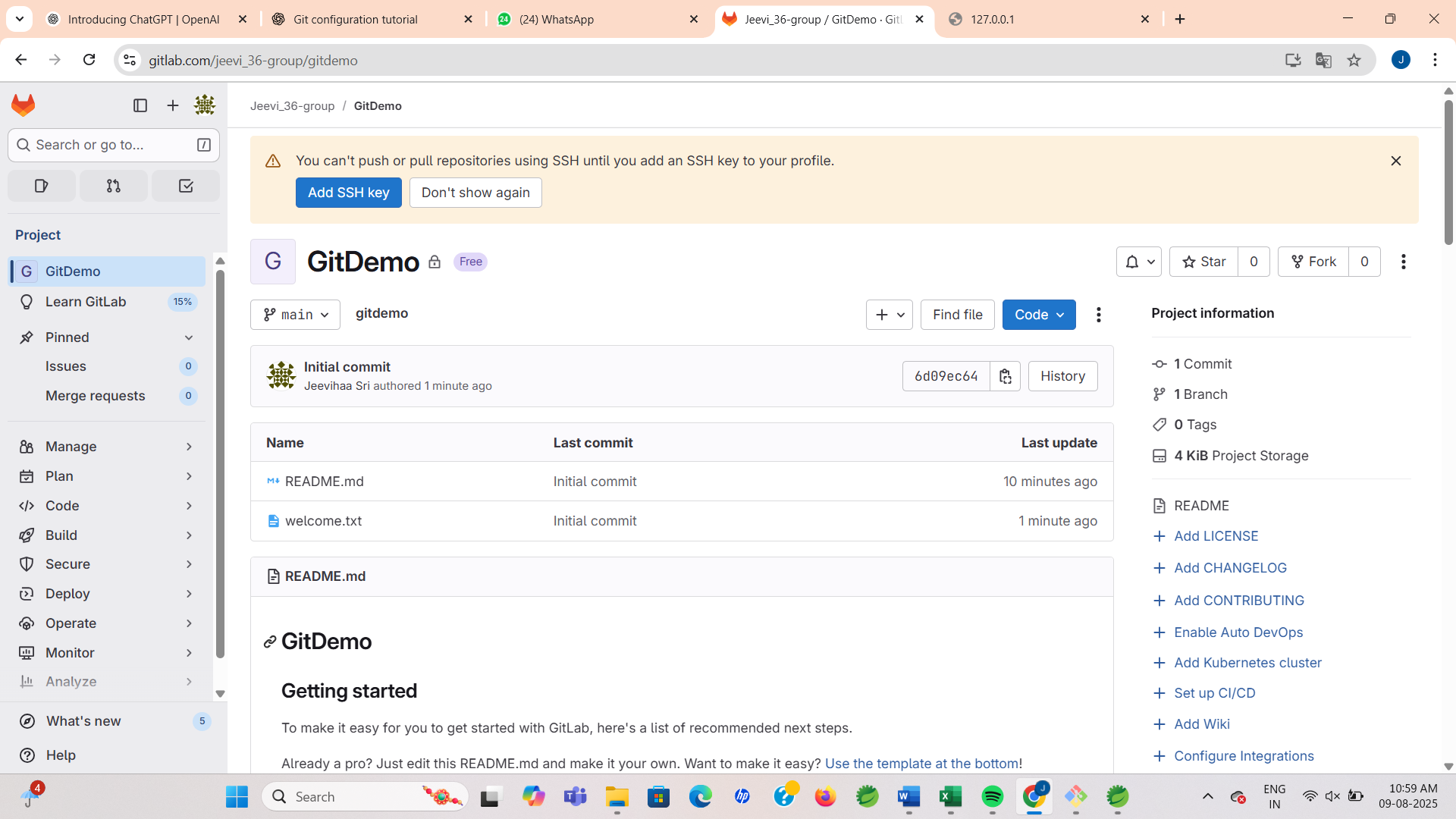
* Create a new repository in GitLab named GitDemo.
* Link local repo to GitLab:



**Step 5: Local repo will sync with GitLab’s default branch:**



**OUTPUT:**

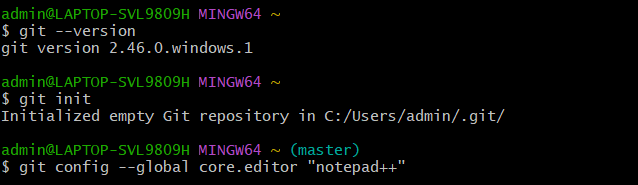


**Exercise 2: Ignoring Unwanted Files in Git Using .gitignore**

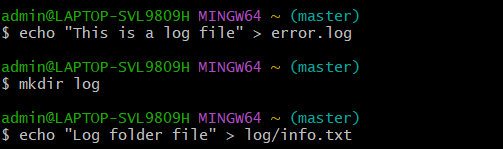
**Scenario:**

You want to prevent all .log files and the entire log folder from being tracked in Git, so you add \*.log and log/ to .gitignore, commit, and push the changes to GitLab.

**Step 1: Pre-requisites**

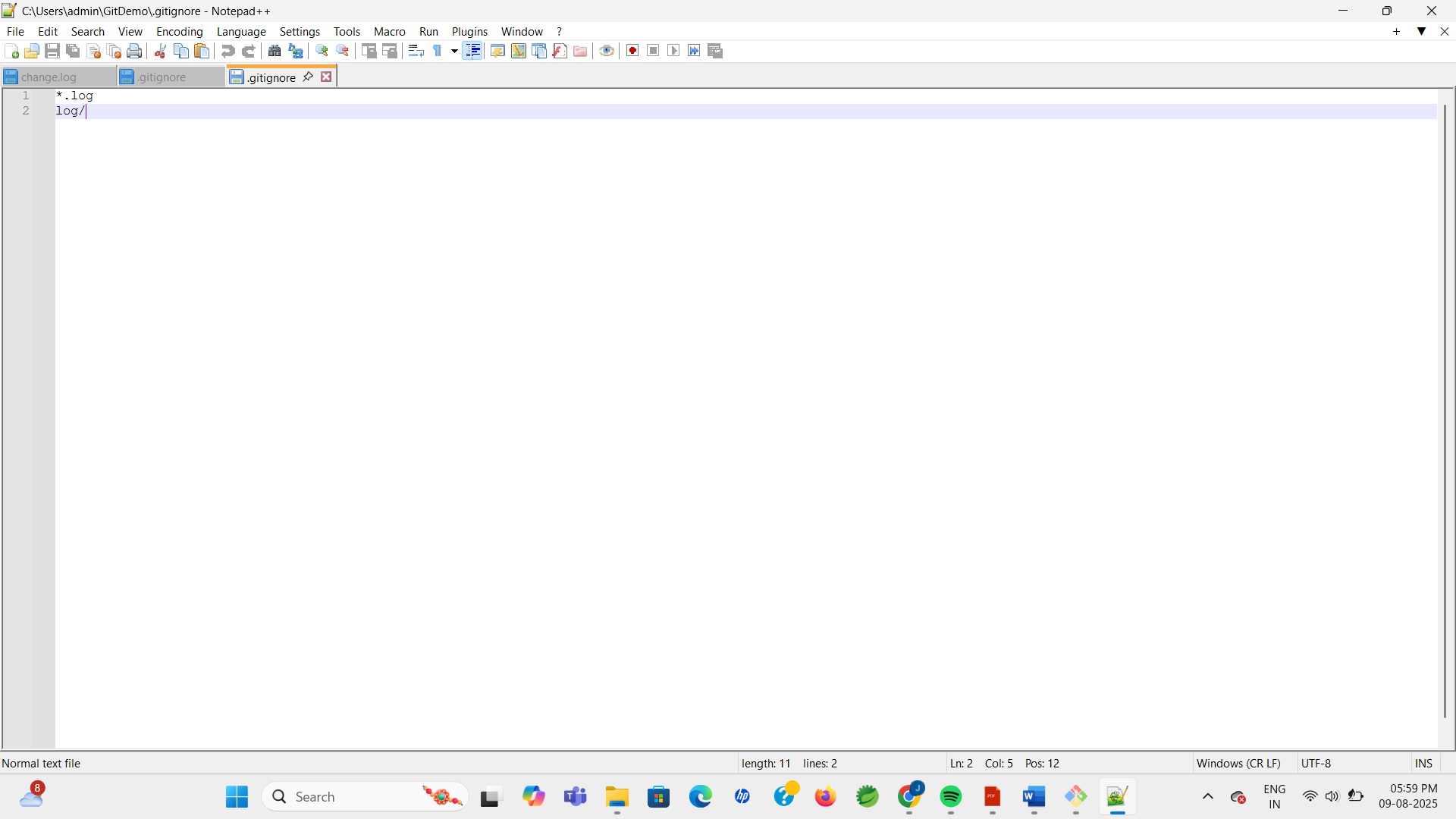


**Step 2: Create Files to Ignore**

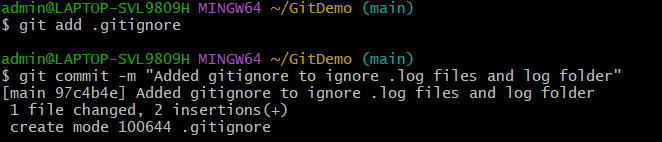
****

**Step 3: Create .gitignore File**

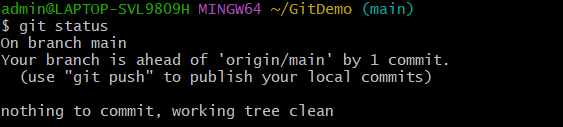
Open .gitignore , then add and save the file



**Step 4: Add .gitignore to Git**

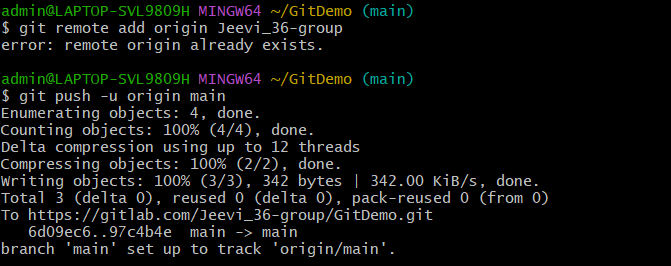
****

**Step 5: Verify Ignored Files**

****

**Step 6: Push to Remote Repository**

**OUTPUT:**

****

**Exercise 3 : Branching and Merging in Git with Merge Requests**

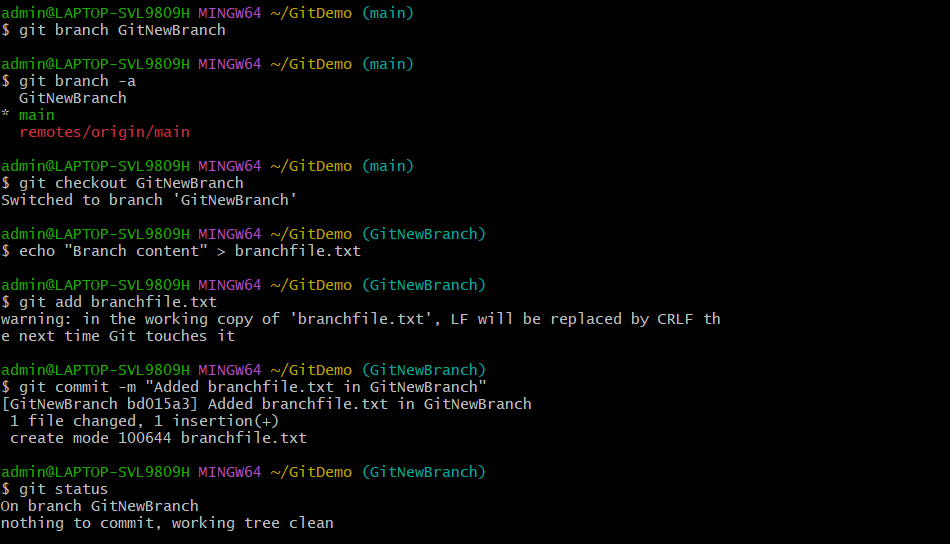
**Scenario:**

You create a new branch to work on changes, then merge it into the master branch and delete the branch after merging.

**Branching:**

* Create a new branch 🡪 List all branches 🡪Switch to the new branch 🡪Add files and contents
* Commit changes and then Check branch status

**OUTPUT:**



**Merging:**

Switch back to master 🡪 Show differences (CLI) 🡪 Show differences (P4Merge tool) 🡪 Merge the branch into master 🡪 View merge history 🡪 Delete the merged branch 🡪 Check status

**OUTPUT:**

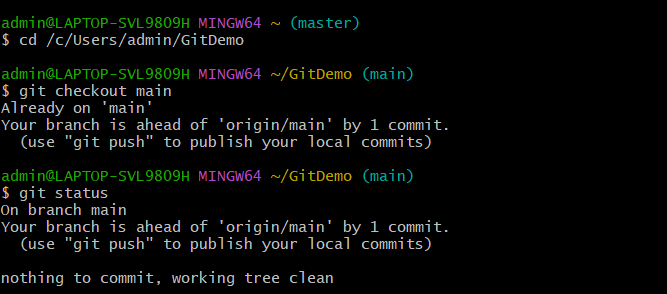


**Exercise 4: Conflict Resolution in Git**

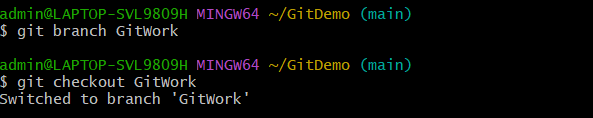
**Scenario:**

You will create a branch (GitWork) and make changes to a file (hello.xml). At the same time, you will make different changes to the same file in the main branch, creating a merge conflict. Then you will resolve the conflict using a 3-way merge, commit, and clean up.

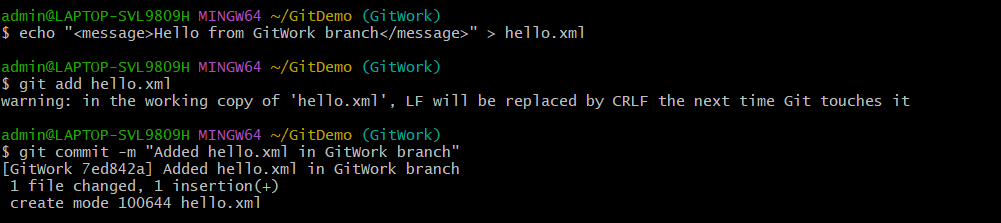
**Step 1: Check main branch is clean**

****

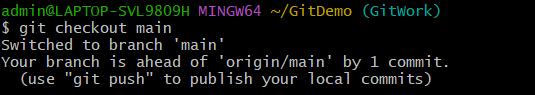
**Step 2: Create and switch to branch**

****

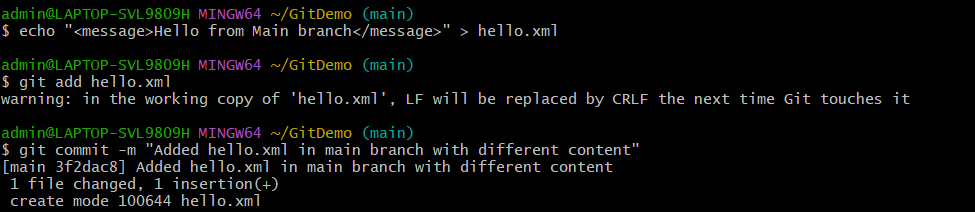
**Step 3: Add file in branch**

****

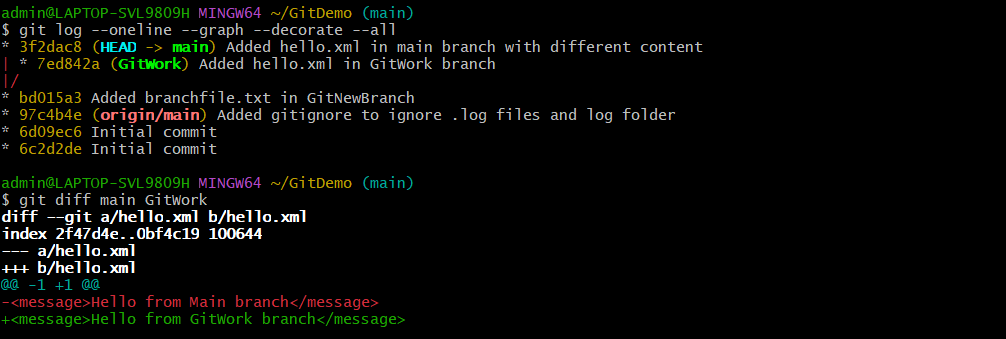
**Step 4: Switch to main branch**

****

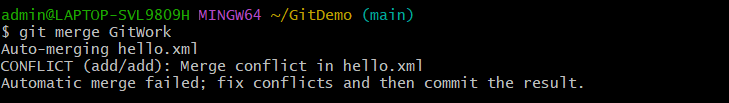
**Step 5: Add different version of same file**

****

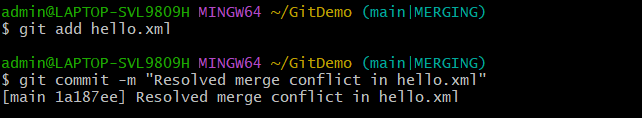
**Step 6: View history & differences**

****

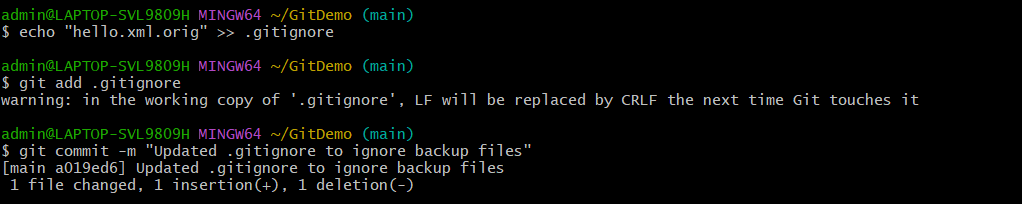
**Step 7: Merge branch into main (expect conflict)**

****

**Step 8: Resolve conflict**

****

**Step 9: Update .gitignore for backup files**

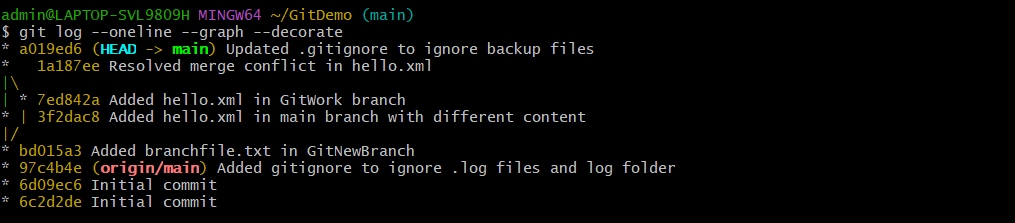
****

**Step 10: Clean up branch**



**Step 11: Final commit history**

**OUTPUT:**

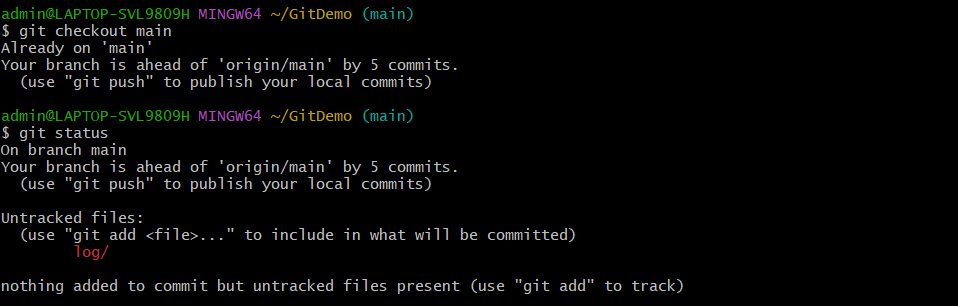
****

**Exercise 5 : Cleaning Up and Pushing Changes to Remote Git Repository**

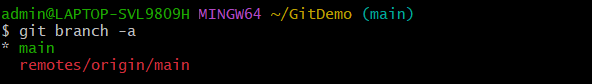
**Scenario:**

Sync local changes with the remote GitLab repository by cleaning up, pulling latest updates, pushing pending commits, and verifying updates in the remote commit history.

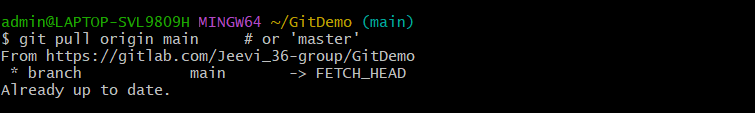
**Step 1 : Verify master (or main) is in clean state**



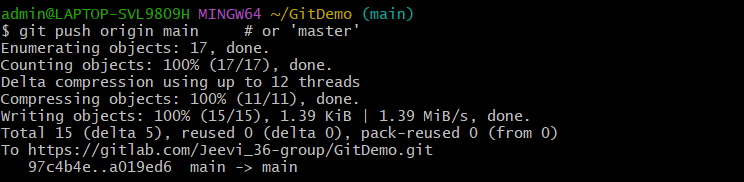
**Step 2 : List all branches**

****

**Step 3 : Pull latest changes from remote into master**



**Step 4: Push pending changes to remote**

****

**Step 5: Verify changes are reflected remotely**

**OUTPUT:**

