# **WEEK 8 git HANDS ON**

**SUBMITTED BY :-**

NAME – AYUSH SAHOO

SUPERSET ID – 6372243

COURSE – DIGITAL NURTURE 4.0 JAVA FSE

EMAIL – [ayush.sahoo003@gmail.com](mailto:ayush.sahoo003@gmail.com)

COLLEGE – C.V. RAMAN GLOBAL UNIVERSITY

**EXERCISE 1:**

**Step 1: Setup your machine with Git Configuration**

**To create a new repository, signup with GitLab and register your credentials**

**Login to GitLab and create a “GitDemo” project**

1. **To check if Git client is installed properly: Open Git bash shell and execute**

**If output shows Git with its version information that indicates, that Git Client installs properly.**

1. **To configure user level configuration of user ID and email ID execute**
2. **To check if the configuration is properly set, execute the following command.**

**Step 2: Integrate notepad++.exe to Git and make it a default editor**

1. **To check, if notepad++.exe execute from Git bash**

**If Git bash could not able to recognize notepad++ command that implies notepad++.exe is note added to the environment path variable.**

**To add path of notepad++.exe to environment variable, go to control panel -> System -> Advanced System settings. Go to Advanced tab -> Environment variables -> Add path of notepad++.exe to the path user variable by clicking on “Edit”**

1. **Exit Git bash shell, open bash shell and execute**

**Now, notepad++ will open from Git bash shell**

1. **To create an alias command for notepad++.exe, execute**

**It will open notepad++ from bash shell, and create a user profile by adding the line in notepad++**

1. **To configure the editor, execute the command**
2. **To verify if notepad++ is the default editor, execute the command**

**Here ‘-e’ option implies editor**

**It will show the entire global configuration as shown below,**

**Step 3: Add a file to source code repository**

1. **Open Git bash shell and create a new project “GitDemo” by executing the command**
2. **Git bash initializes the “GitDemo” repository. To verify, execute the command**

**It will display all the hidden files in the Git “working directory”.**

1. **To create a file “welcome.txt” and add content to the file, execute the command**
2. **To verify if the file “welcome.txt” is created, execute**
3. **To verify the content, execute the command**
4. **Check the status by executing**

**Now the file “welcome.txt” is available in Git “working directory”**

1. **To make the file to be tracked by Git repository, execute the command**
2. **To add multi line comments, we are opening default editor to comment. Execute the command**

**Notepad++ editor will open and to add multi-line comment with default editor**

1. **To check if local and “Working Directory” git repository are same, execute git status**

**welcome.txt is added to the local repository.**

1. **Signup with GitLab and create a remote repository “GitDemo”**
2. **To pull the remote repository, execute**

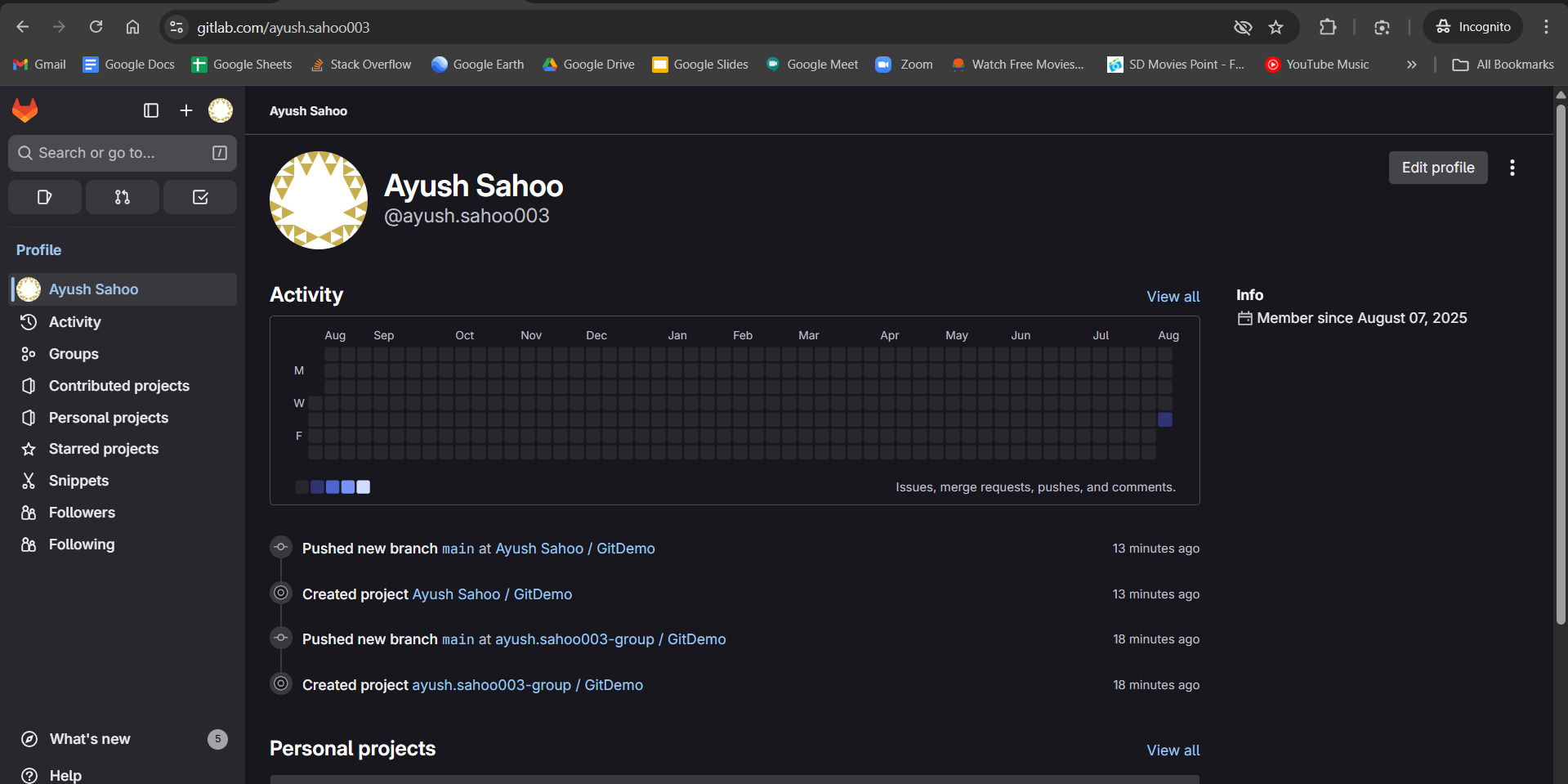
**git pull origin master**

1. **To push the local to remote repository, execute**

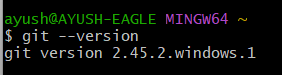
**git push origin master**

**SOLUTION:**

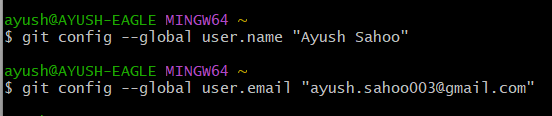
Step 1: GitLab account made with personal credentials.



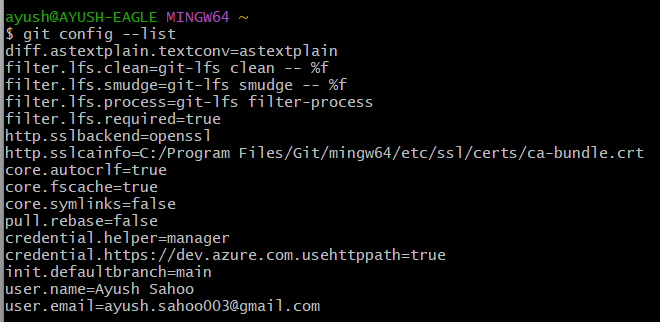
Step 2: Check Git version.



Step 3: Configured user level credentials by entering user name and user email.



Step 4: Whether the user name and user email has been properly inputted or not.



Step 5: Notepad++ working fine.

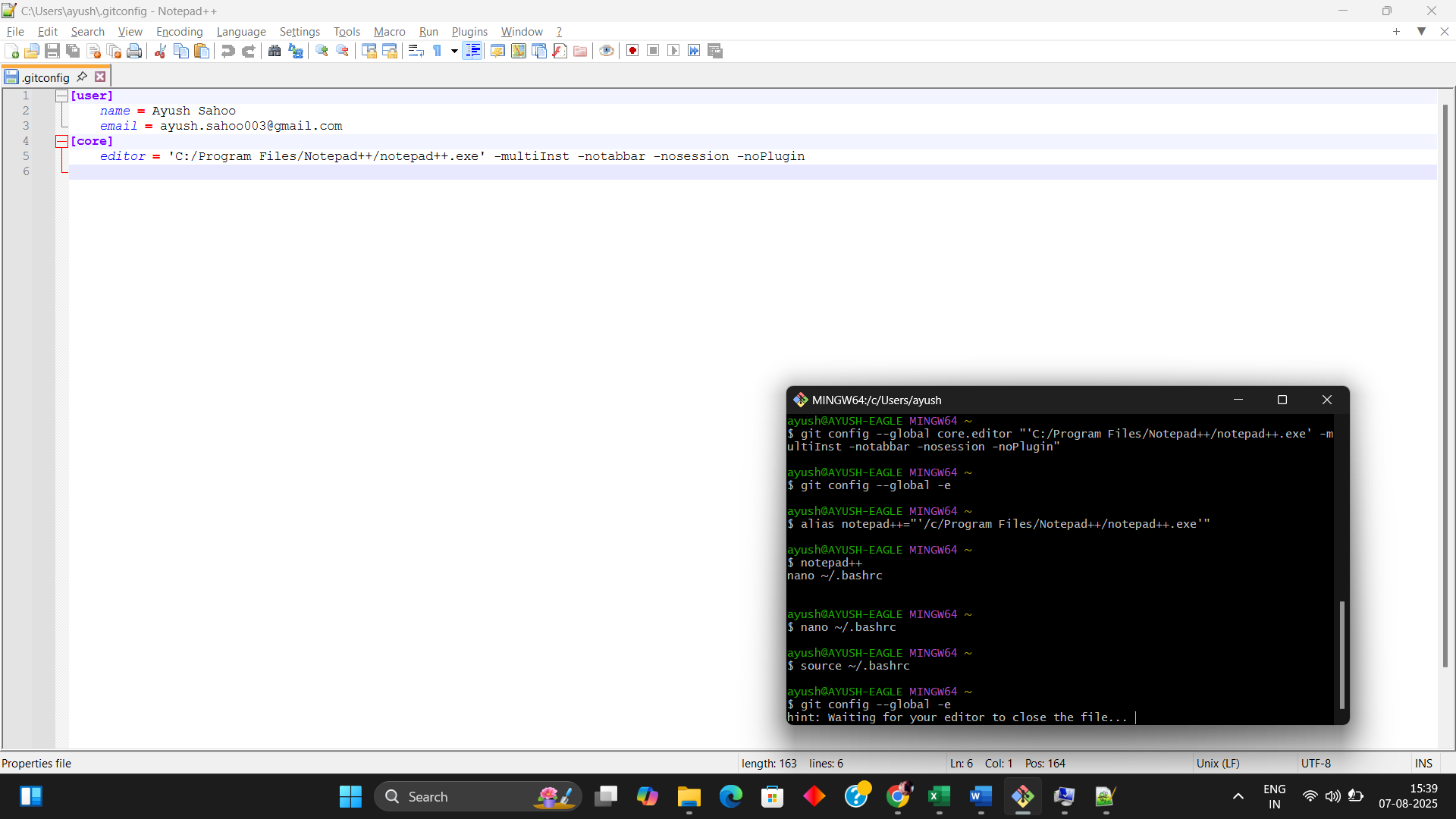


Step 6: For Alias.

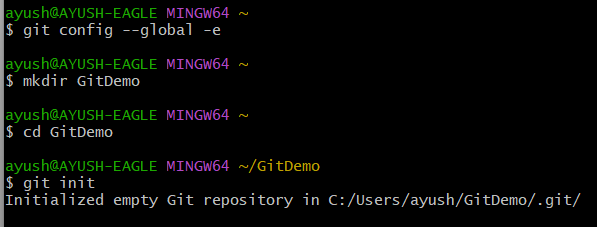


Step 7:

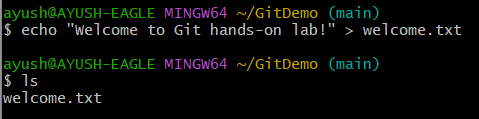


****

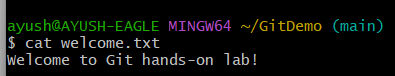
Step 8: For the demo folder.



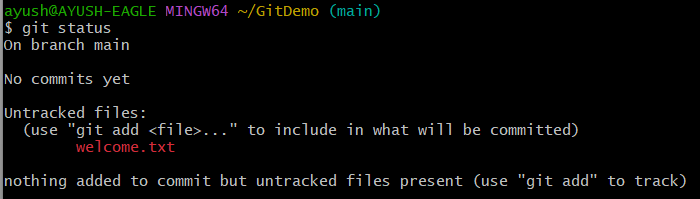
Step 9: Create a “welcome.txt” and to check it whether the folder is available or not.



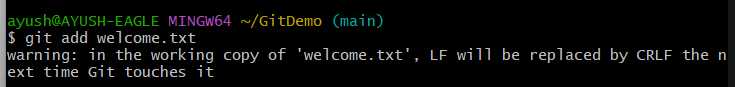
Step 10: to verify what’s written in “welcome.txt”.



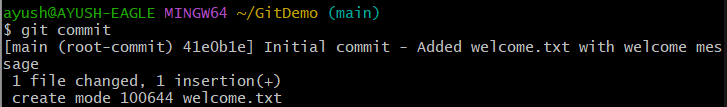
Step 11: Checking the Status.



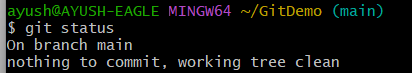
Step 12: To make the file tracked by git repository.



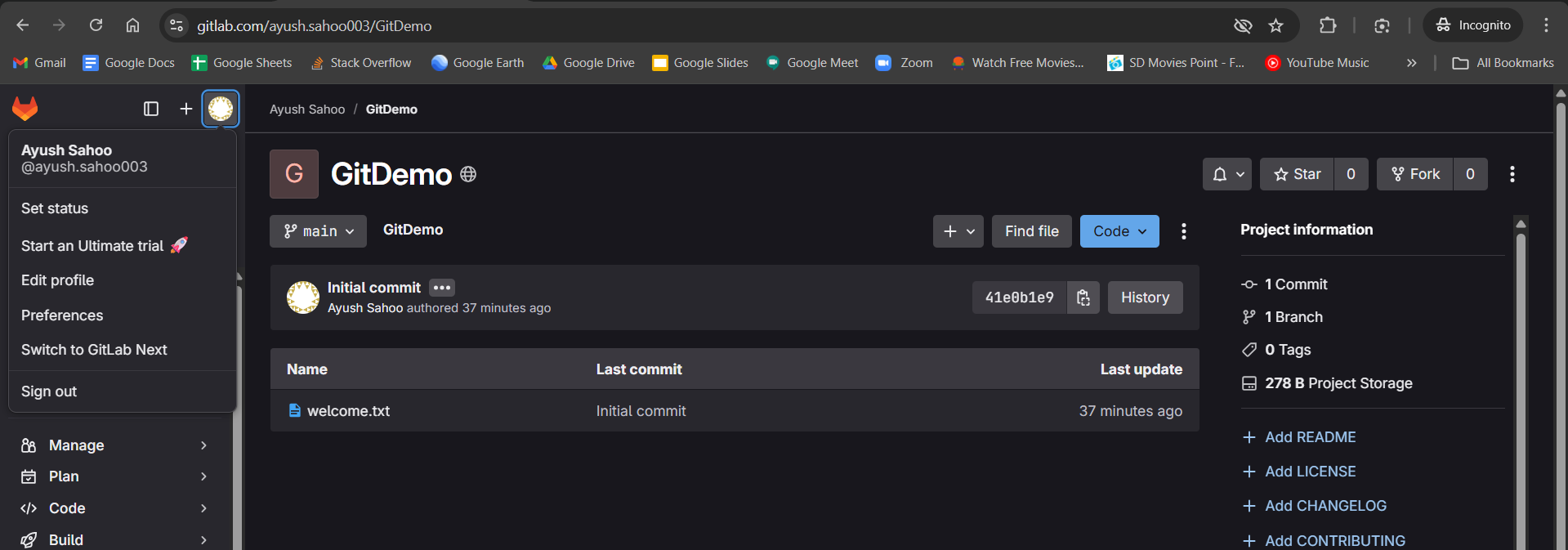
Step 13: To add multi-lines comments (It will actually open the notepad++).



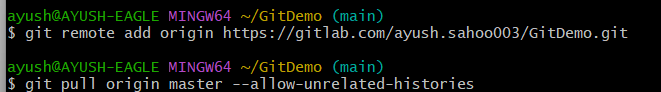
Step 14: Checking is local and “Working directory” git repository are same.



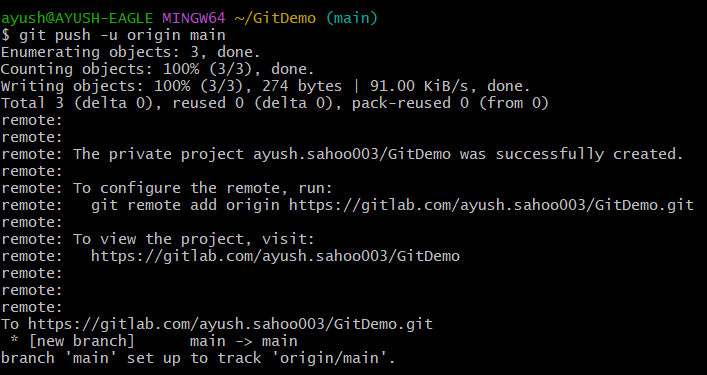
Step 15: Create a remote repository “GitDemo” in GitLab.



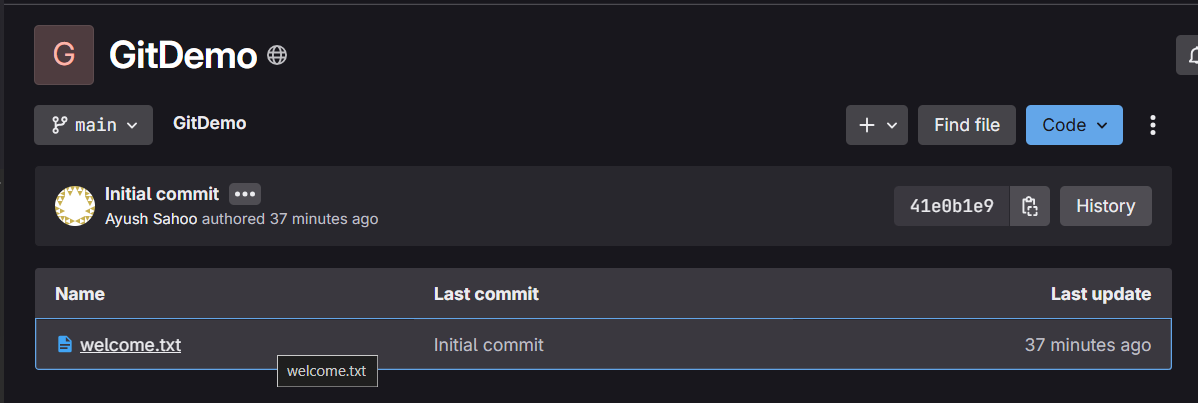
Step 16: To Pull the remote repository.



Step 17: To Push the remote repository.



OUTPUT:



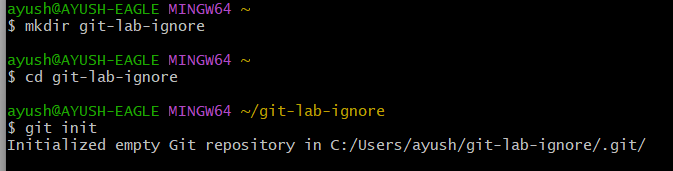
**EXERCISE 2:**

**Create a “.log” file and a log folder in the working directory of Git. Update the .gitignore file in such a way that on committing, these files (.log extensions and log folders) are ignored.**

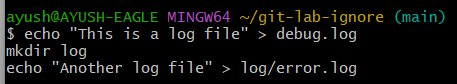
**Verify if the git status reflects the same about working directory, local repository and git repository.**

**SOLUTION:**

Step 1: Create another file by the name of “git-lab-ignore”



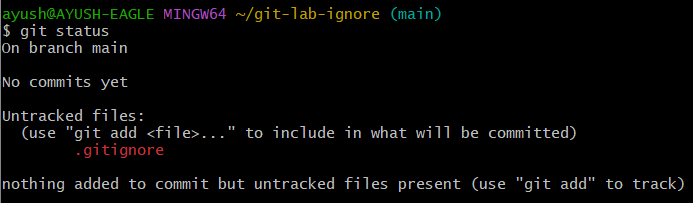
Step 2: log files.



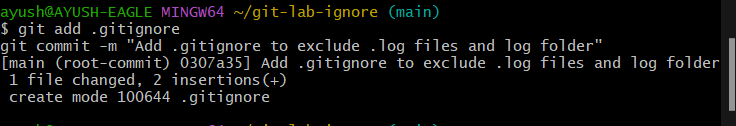
Step 3: Opening notepad++ for editing files.

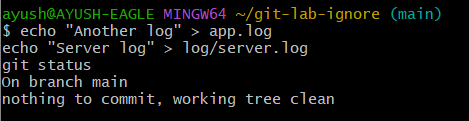


Step 4: checking git status.



Step 4: git and git ignore

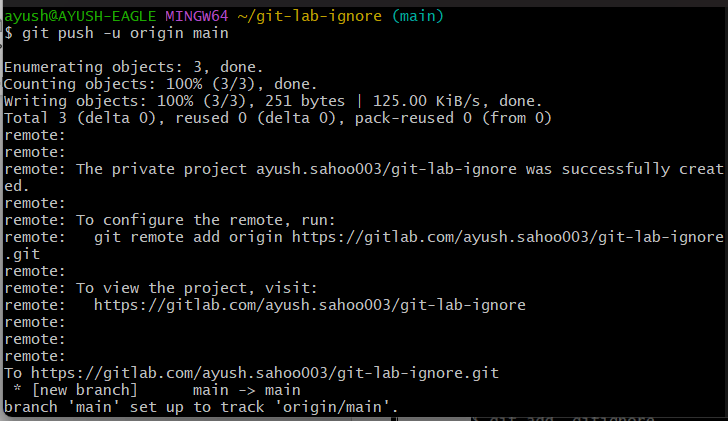




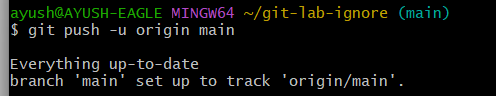
Step 5: Adding files remotely to GitLab.



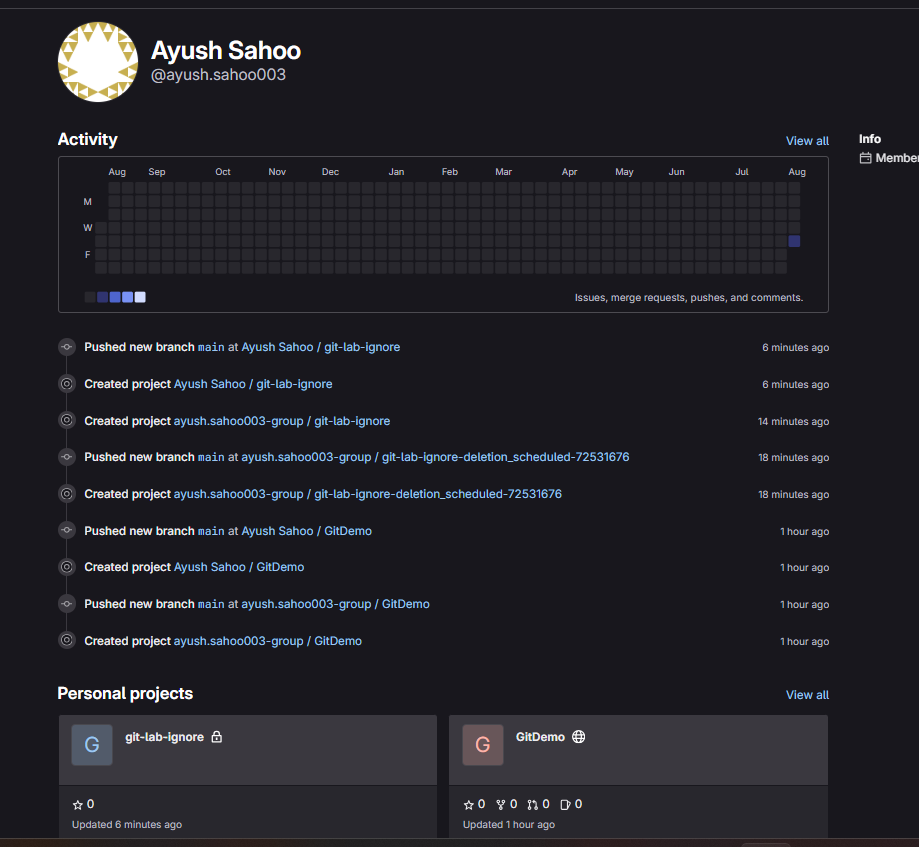
Step 6: Pushing content to GitLab.



Step 7: Pulling request for GitLab for file named “git-lab-ignore”



OUTPUT:



**EXERCISE 3:**

**Please follow the instruction to complete the hands-on. Each instruction expects a command for the Git Bash.**

**Branching:**

1. **Create a new branch “GitNewBranch”.**
2. **List all the local and remote branches available in the current trunk. Observe the “\*” mark which denote the current pointing branch.**
3. **Switch to the newly created branch. Add some files to it with some contents.**
4. **Commit the changes to the branch.**
5. **Check the status with “git status” command.**

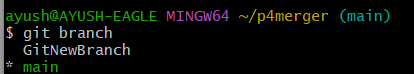
**Merging:**

1. **Switch to the master**
2. **List out all the differences between trunk and branch. These provide the differences in command line interface.**
3. **List out all the visual differences between master and branch using P4Merge tool.**
4. **Merge the source branch to the trunk.**
5. **Observe the logging after merging using “git log –oneline –graph –decorate”**
6. **Delete the branch after merging with the trunk and observe the git status.**

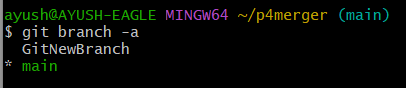
**SOLUTION:**

Step 1:Created a new branch by name of “GitNewBranch”

(I exited the app that why I don’t have a screenshot of that while creating a new branch but here is the proof that I did it)



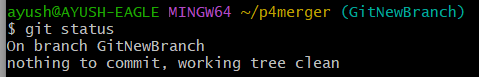
Step 2:



Step 3: Switch to new branch

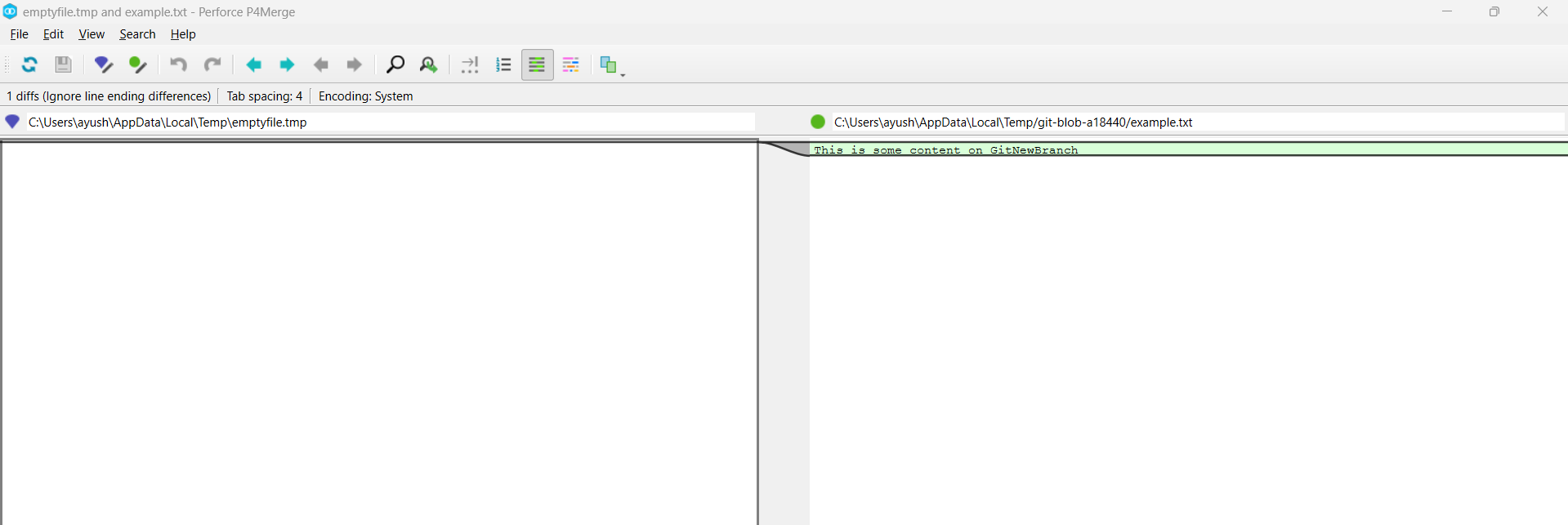


Step 4: Add a file with content and commit and checking the git status

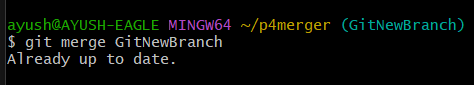


Step 5: List differences between main and GitNewBranch.

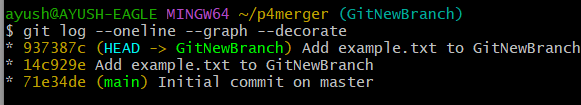




Step 6: Merger new and GitNewBranch.



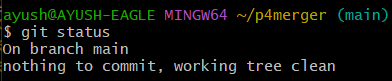
Step 7: Observe merch log.



Step 8: Delete branch after merging



Step 10: Check status



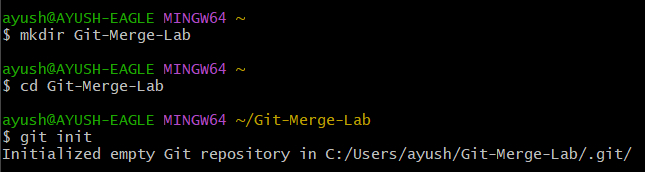
**EXERCISE 4:**

**Please follow the instructions to complete the hands-on. Each instruction expect a command for the Git Bash.**

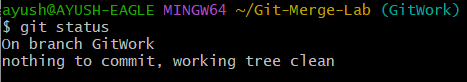
1. **Verify if master is in clean state.**
2. **Create a branch “GitWork”. Add a file “hello.xml”.**
3. **Update the content of “hello.xml” and observe the status**
4. **Commit the changes to reflect in the branch**
5. **Switch to master.**
6. **Add a file “hello.xml” to the master and add some different content than previous.**
7. **Commit the changes to the master**
8. **Observe the log by executing “git log –oneline –graph –decorate –all”**
9. **Check the differences with Git diff tool**
10. **For better visualization, use P4Merge tool to list out all the differences between master and branch**
11. **Merge the bran to the master**
12. **Observe the git mark up.**
13. **Use 3-way merge tool to resolve the conflict**
14. **Commit the changes to the master, once done with conflict**
15. **Observe the git status and add backup file to the .gitignore file.**
16. **Commit the changes to the .gitignore**
17. **List out all the available branches**
18. **Delete the branch, which merge to master.**
19. **Observe the log by executing “git log –oneline –graph –decorate”**

**SOLUTION:**

Step 1: Made a another repo for this question by the name of “Git-Manage-Lab”

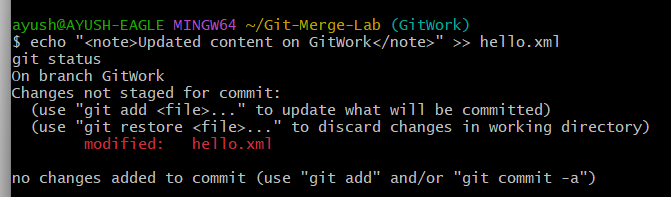


Step 2: verifying whether it is a clean state or not

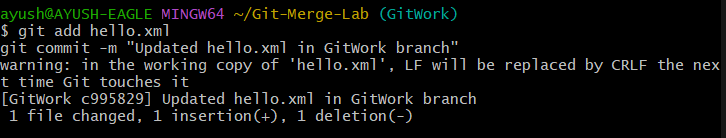


**(NOTE: HERE I DID ALREADY MADE THIS BRANCH FROM MAIN TO GITWORK AND ADDED A FILE NAME “hello.xml”)**

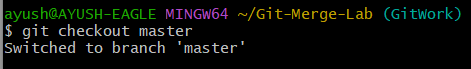
Step 3: Updated the content of hello.xml



Step 4: Committed the changes to reflect in branch.



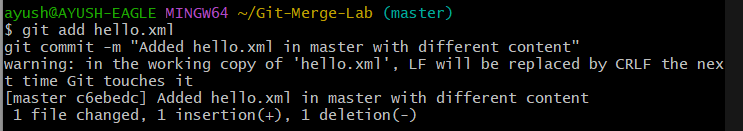
Step 5: Switched to master.



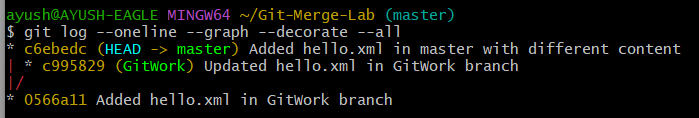
Step 6: Added different content to file “hello.xml” in master branch.



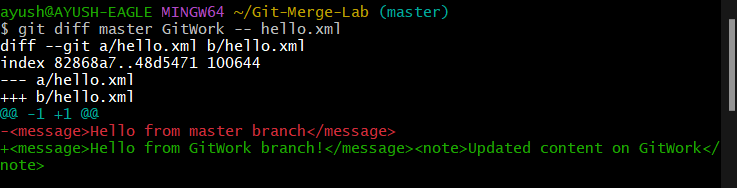
Step 7: Committed the changes in master branch.



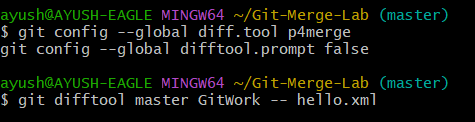
Step 8: Observing the log by executing “git log –online –graph –decorate –all”

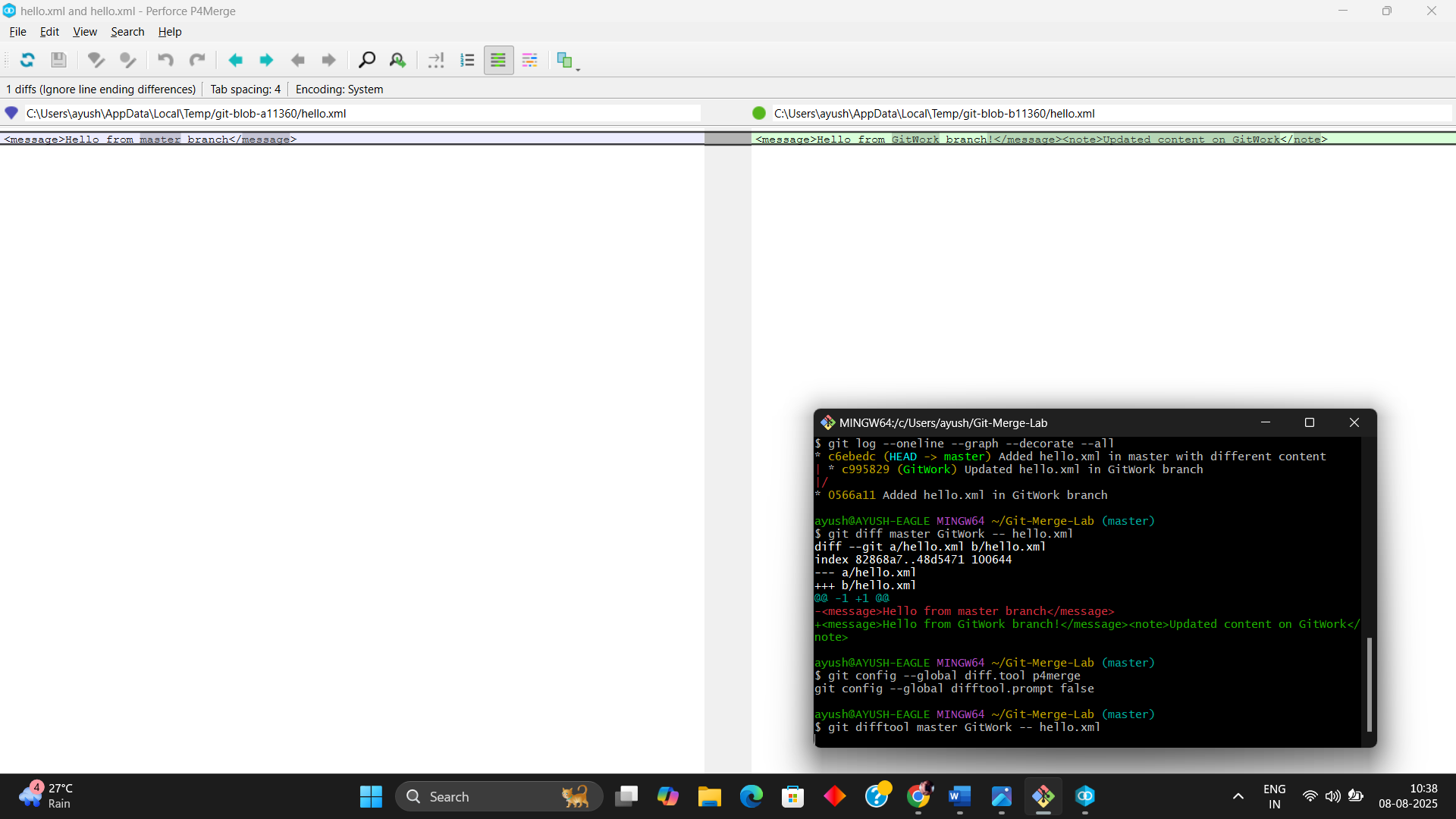


Step 9: Checking the difference with git diff tool.

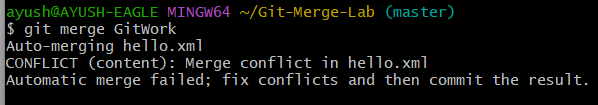


Step 10: Use P4Merger to see all the differences

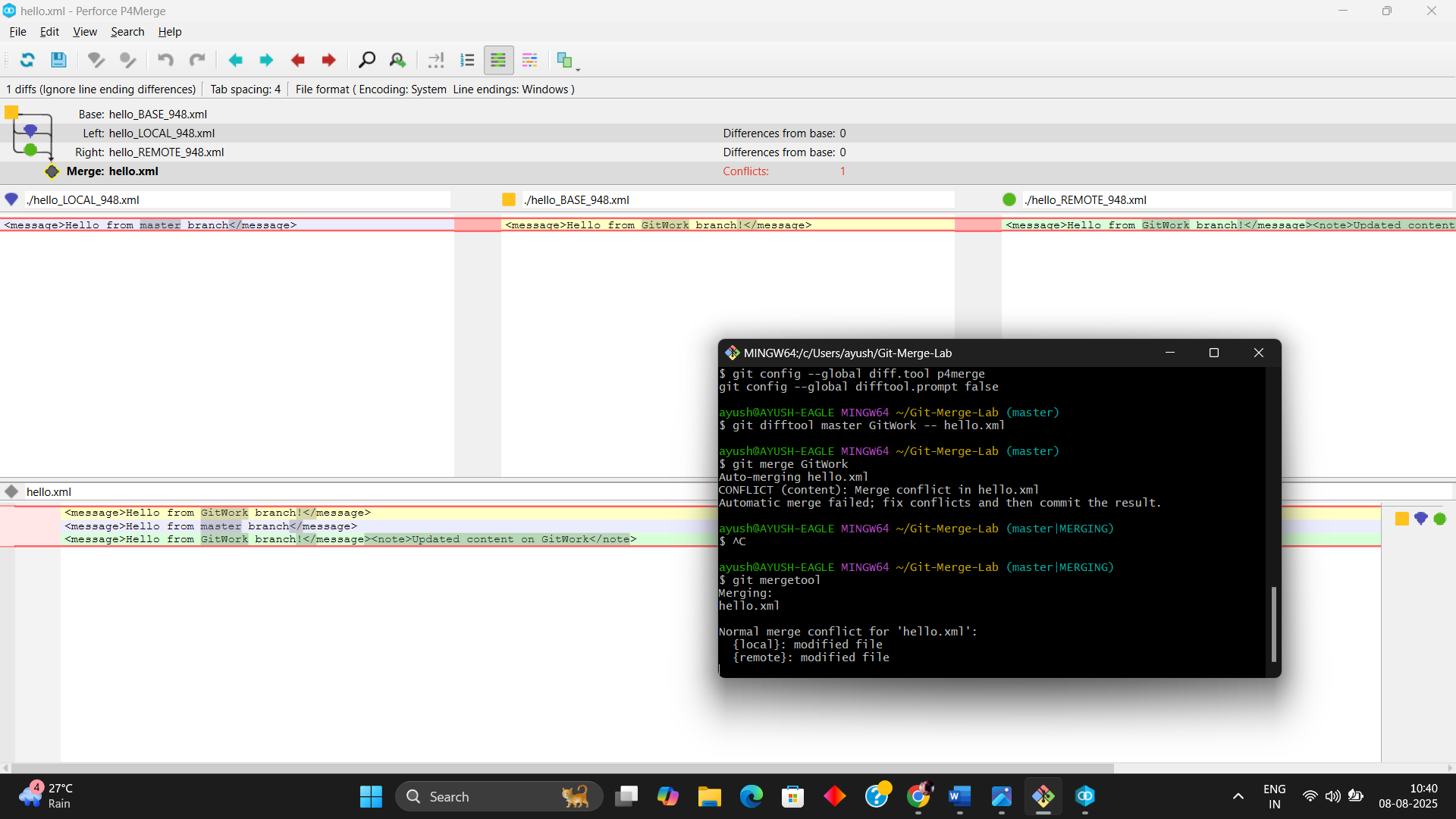




Step 11: Merge the GitWork and master branches



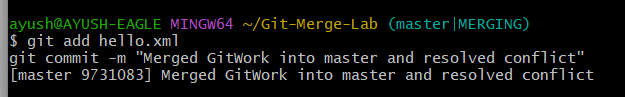
**(NOTE: IT WILL ALWAYS SHOW THE ERROR BUT THERE TO SOLVE IT YOU NEED TO UPDATE THIS IN P4MERGER AND SAVE THE FILE AND CLOSE IT WILL BE FIXED)**



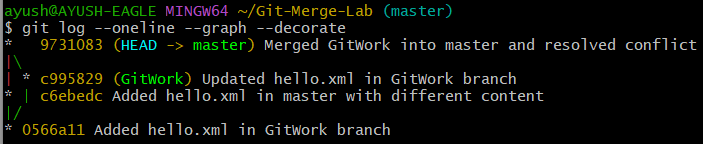
STEP 12: merging the git



Step 13: Conflict resolved



Step 14: Executing the “git log –online –graph –decorate.



**EXERCISE 5:**

**Please follow the instructions to complete the hands-on. Each instruction expects a command for the Git Bash.**

1. **Verify if master is in clean state.**
2. **List out all the available branches.**
3. **Pull the remote git repository to the master**
4. **Push the changes, which are pending from “Git-T03-HOL\_002” to the remote repository.**
5. **Observe if the changes are reflected in the remote repository.**

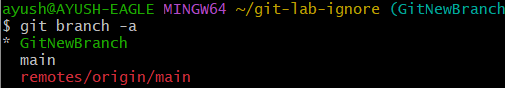
**SOLUTION:**

Step 1: Verify the clean state for the repo..

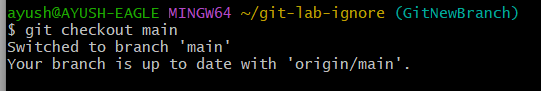
(NOTE: I HAVE USED THE SAME REPO USED IN 2ND ASSIGNMENT)



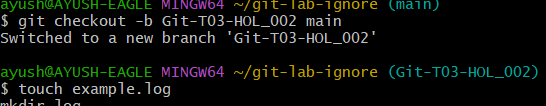
Step 2: List all the branches

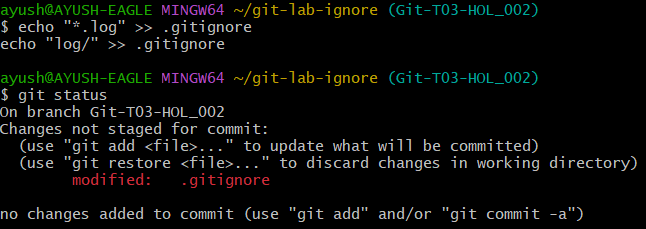


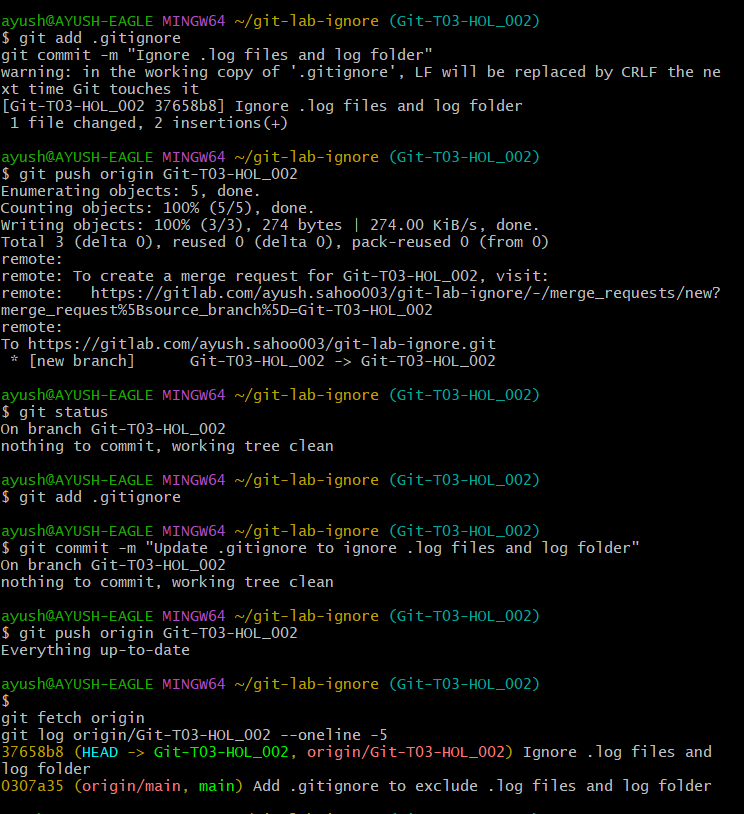
Step 3: Pulled the repository in the main (not master as it is easy for me to work and I have done this like this is every other assignments).



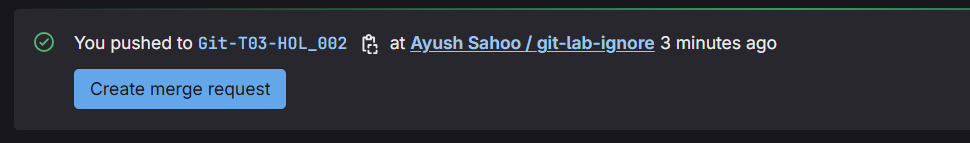
Step 4: Pushed the changes by making another branch with the name of “**Git-T03-HOL\_002**”.

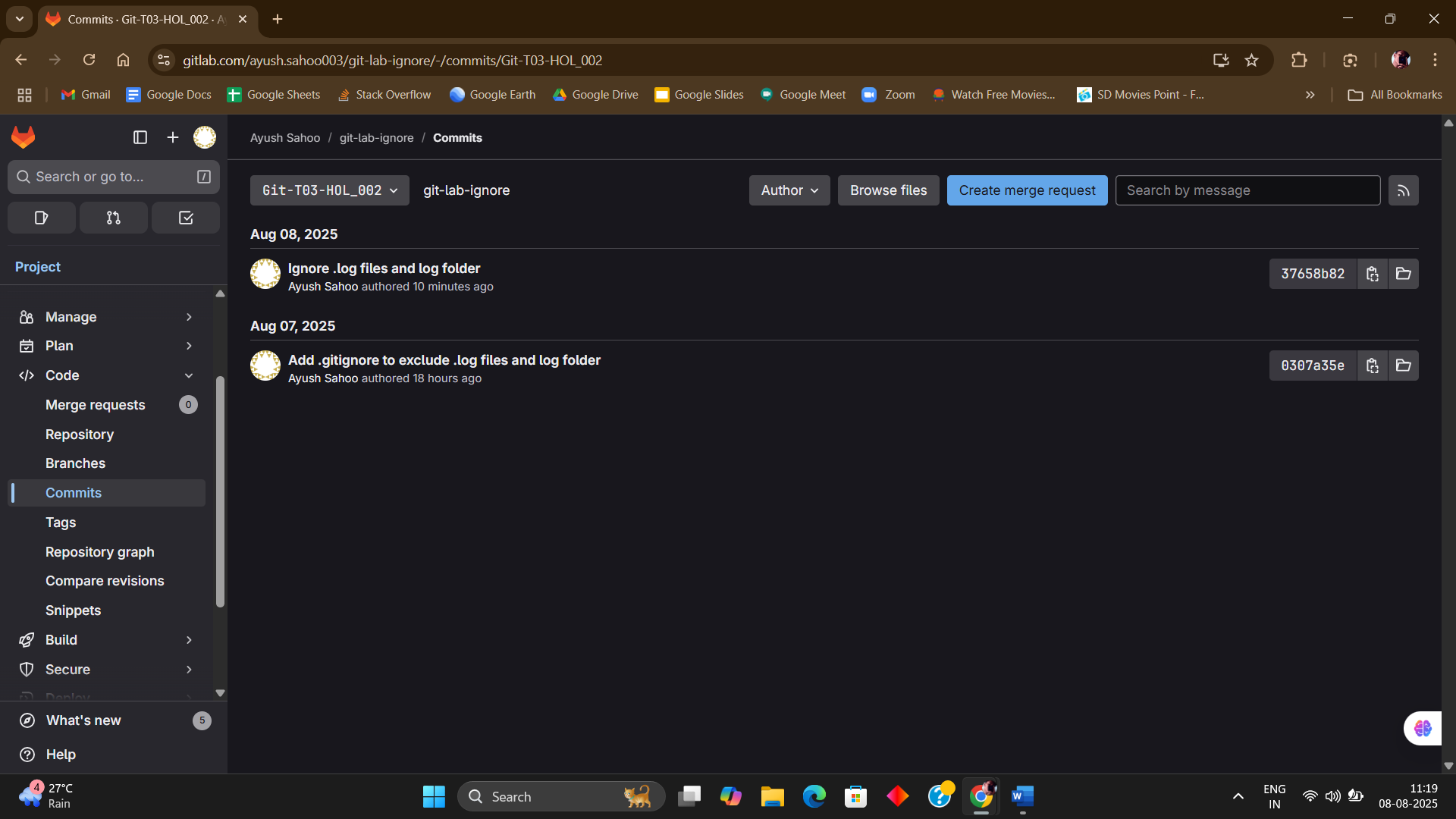






Step 5: confirming by checking the remote (GitLab here)





**x----x----x**