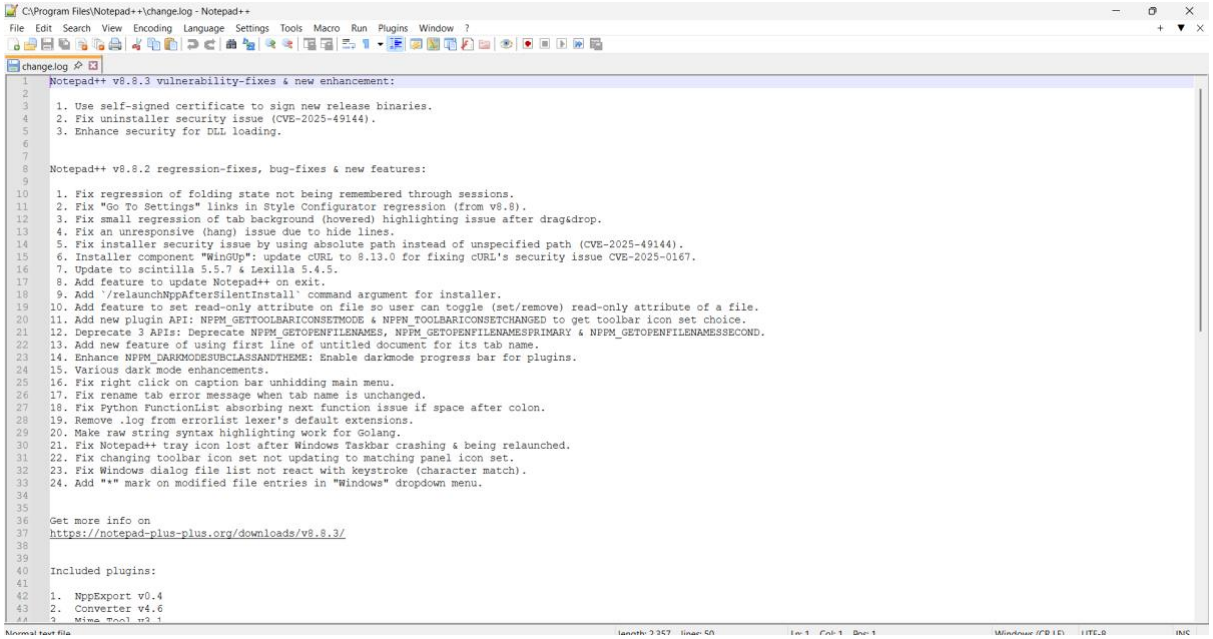
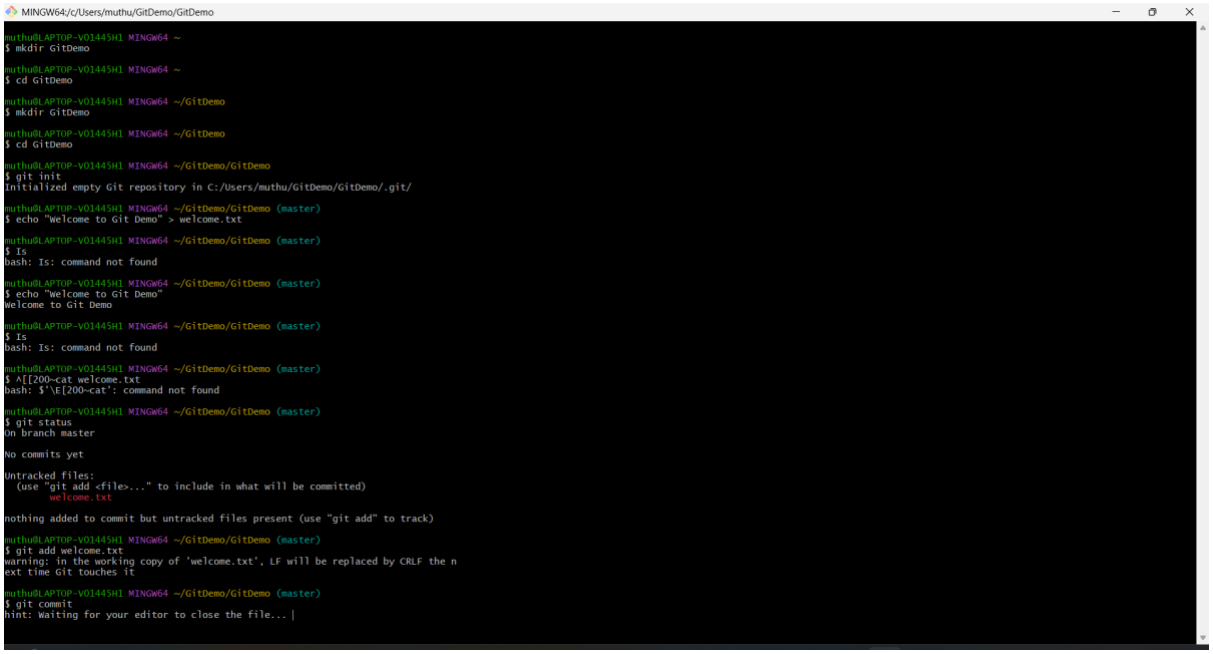
WEEK 8 – GIT

1.In this hands-on lab, you will learn how to

- Setup your machine with Git Configuration
- Integrate notepad++.exe to Git and make it a default editor
- Add a file to source code repository

OUTPUT:





2. Objectives

- Explain git ignore
- Explain how to ignore unwanted files using git ignore

In this hands-on lab, you will learn how to:

- Implement git ignore command to ignore unwanted files and folders

Prerequisites

The following are the pre-requisites to complete this hands-on lab:

- Setting up Git environment
- Integrate notepad++ as a default editor
- A Git repository in the local system and a remote repository in GitLab
- Estimated time to complete this lab: **20 minutes.**
- 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.
-

Open **Git Bash** and run:

cd GitDemo

If not created yet:

mkdir GitDemo

cd GitDemo
git init

---

2. Create unwanted files/folders
echo "This is a log file" > error.log
mkdir log
echo "Log file in folder" > log/app.log

This creates:

- A file: error.log
- A folder: log/ containing app.log

---

3. Check Git status before ignoring
git status
Untracked files:
  error.log
  log/

✅ 4. Create a .gitignore file
notepad++ .gitignore

Add the following lines:

*.log

log/

Explanation:

- *.log → Ignore all .log files
- log/ → Ignore the entire log folder

Save and close the file.

---

5. Check Git status again
git status

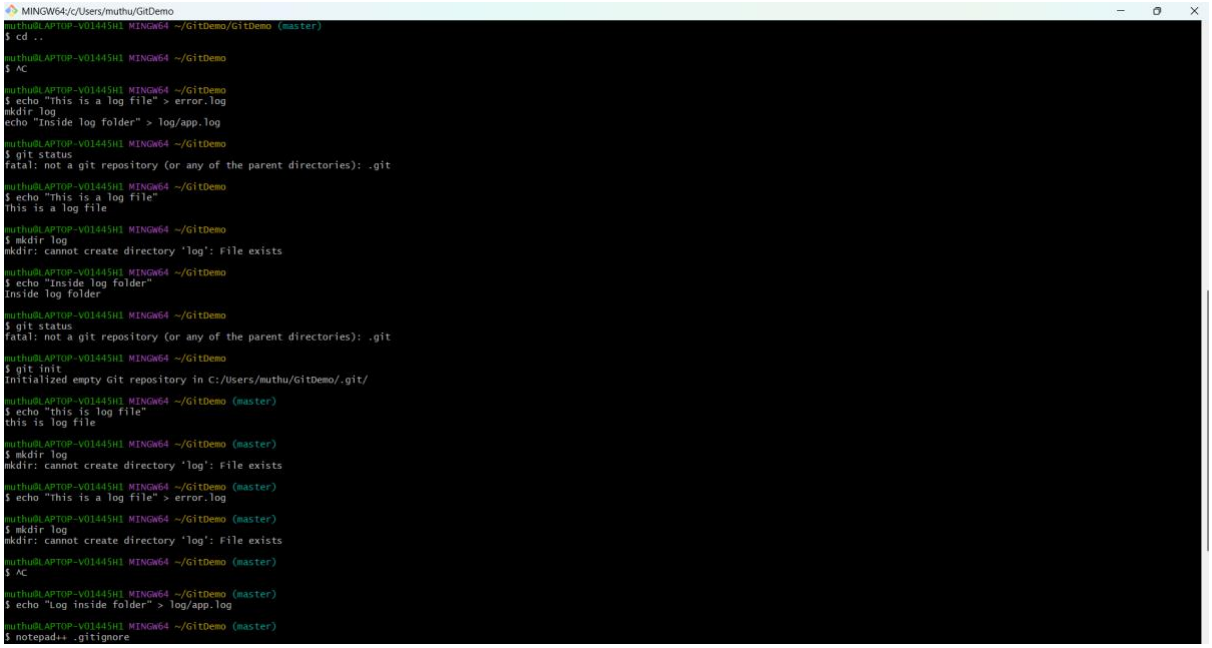
---

6. Add and commit .gitignore only
git commit -m "Added .gitignore to ignore .log files and log folder"

---

7. Push to GitLab (if connected)
git push origin master

Use this only if your Git project is linked to a remote GitLab repo.



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

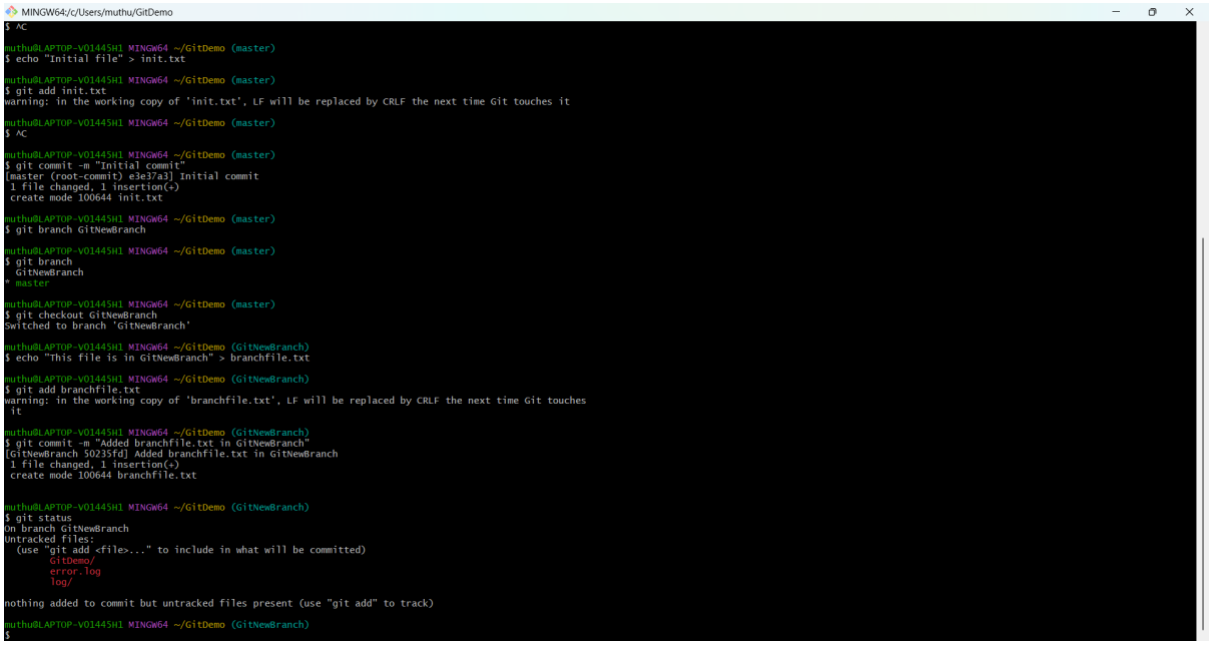
Branching:

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.

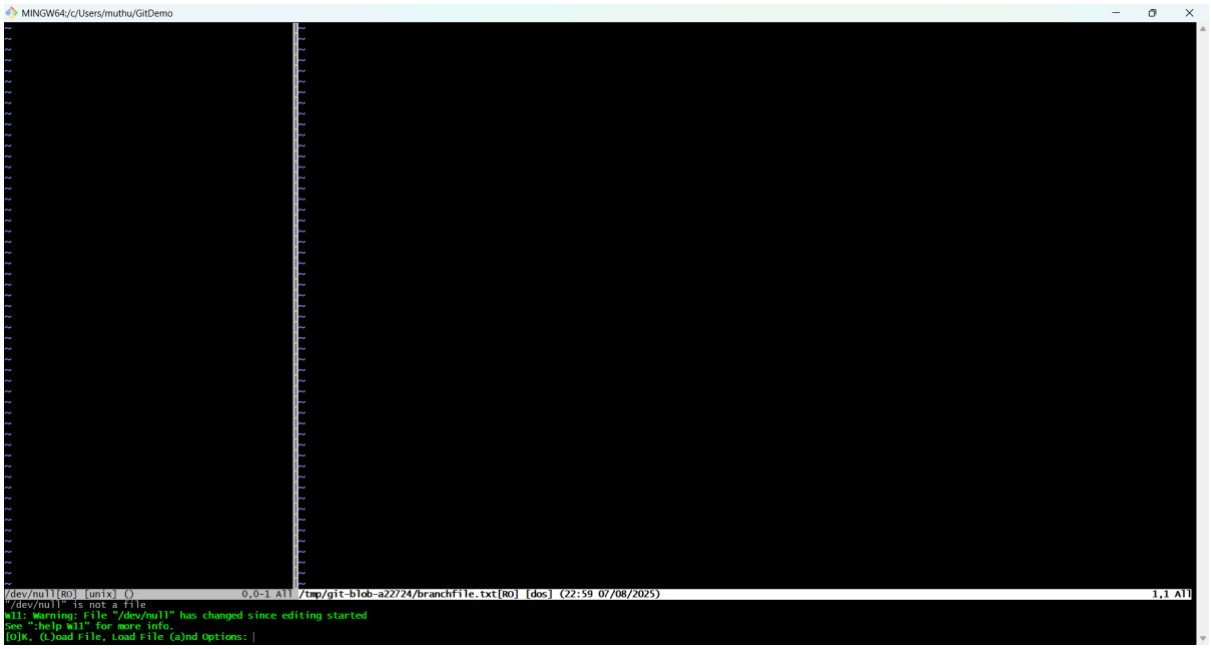
Merging:

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.

output for branching:



For merging:



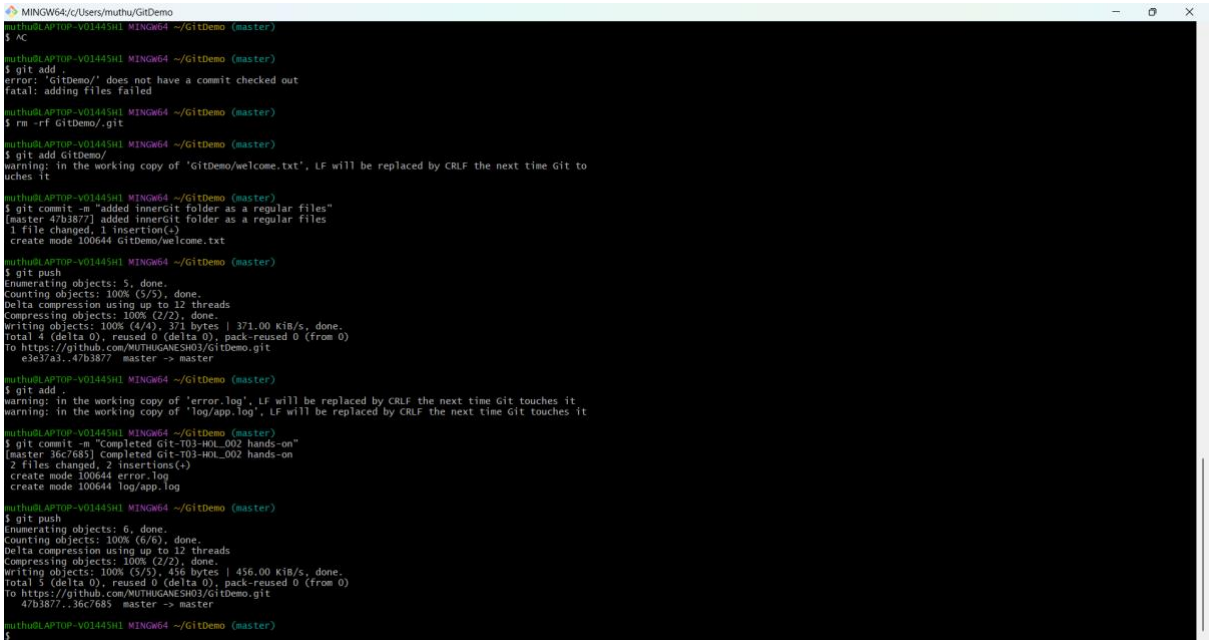
Estimated time to complete this lab: **10 minutes.**

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.

Output:



In github: