**COGNIZANT - DIGITALNURTURE4.0**

**DEEPSKILLING JAVA FSE**

**WEEK 08: GIT**

**Exercise 1:**

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

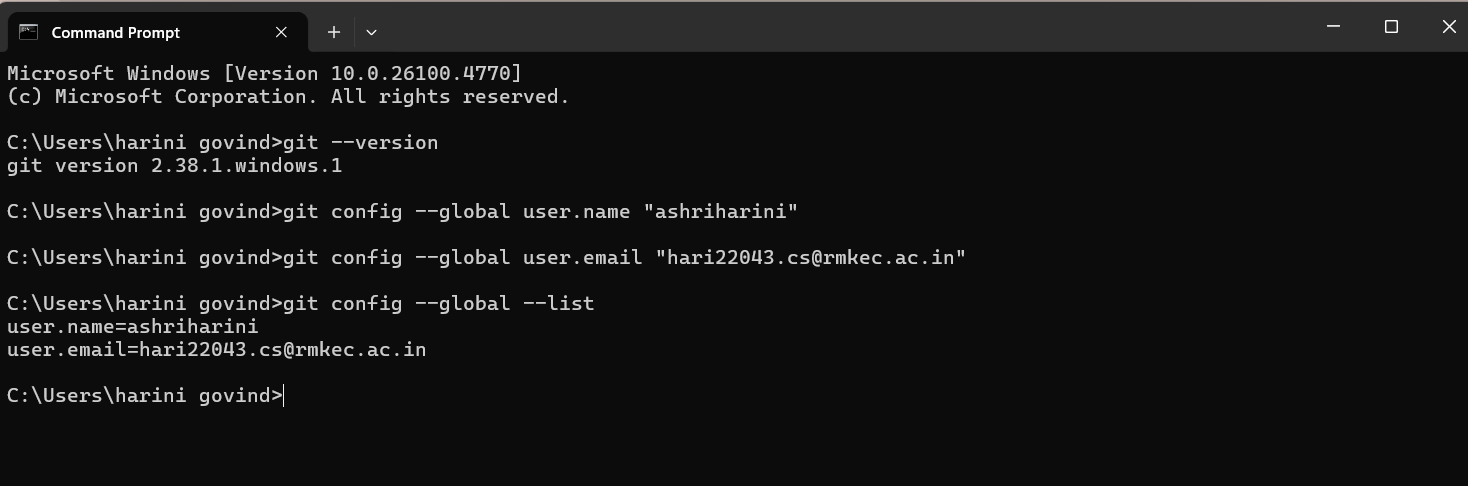
This step verifies Git installation, sets up the global username and email, and confirms the configuration.

git –version

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

git config --global user.email "you@example.com"

git config --global –list

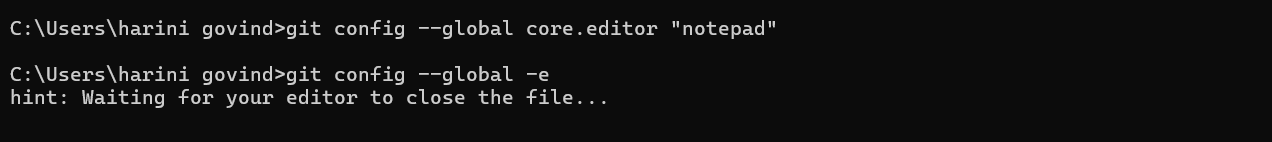


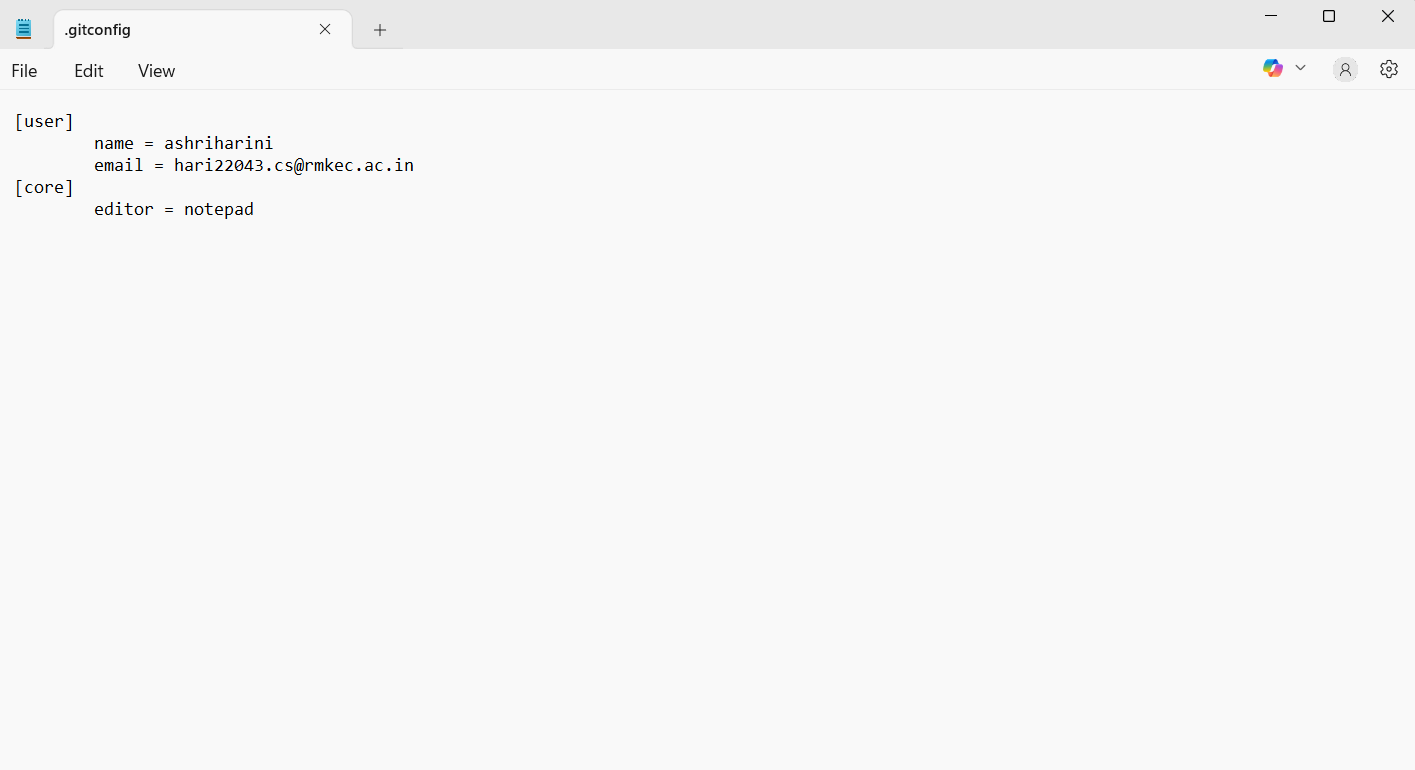
**Step 2: Integrate notepad++.exe to Git and make it a default editor**

This step configures Notepad as the default text editor for commit messages and Git operations.

git config --global core.editor "notepad"

git config --global -e

****

****

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

This step creates a local repository, adds a file, stages it, and commits it.

mkdir GitDemo

cd GitDemo

git init

echo Welcome to Git Demo > welcome.txt

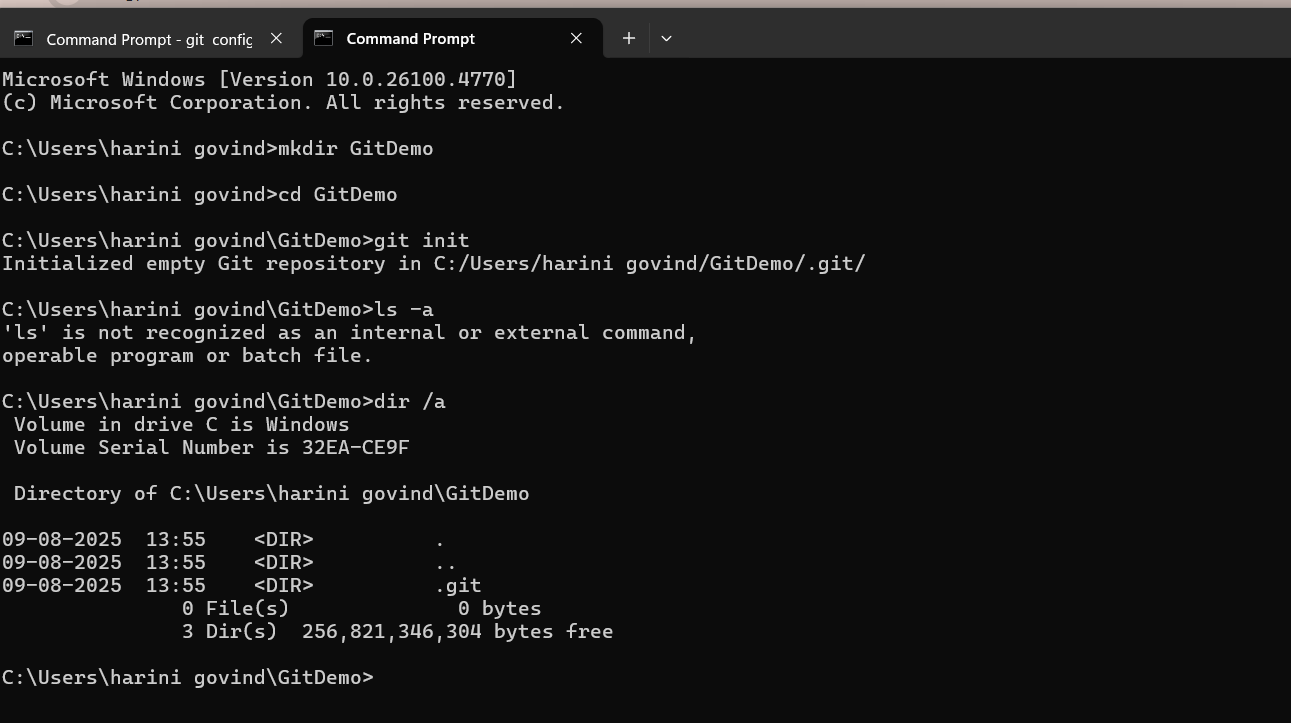
type welcome.txt

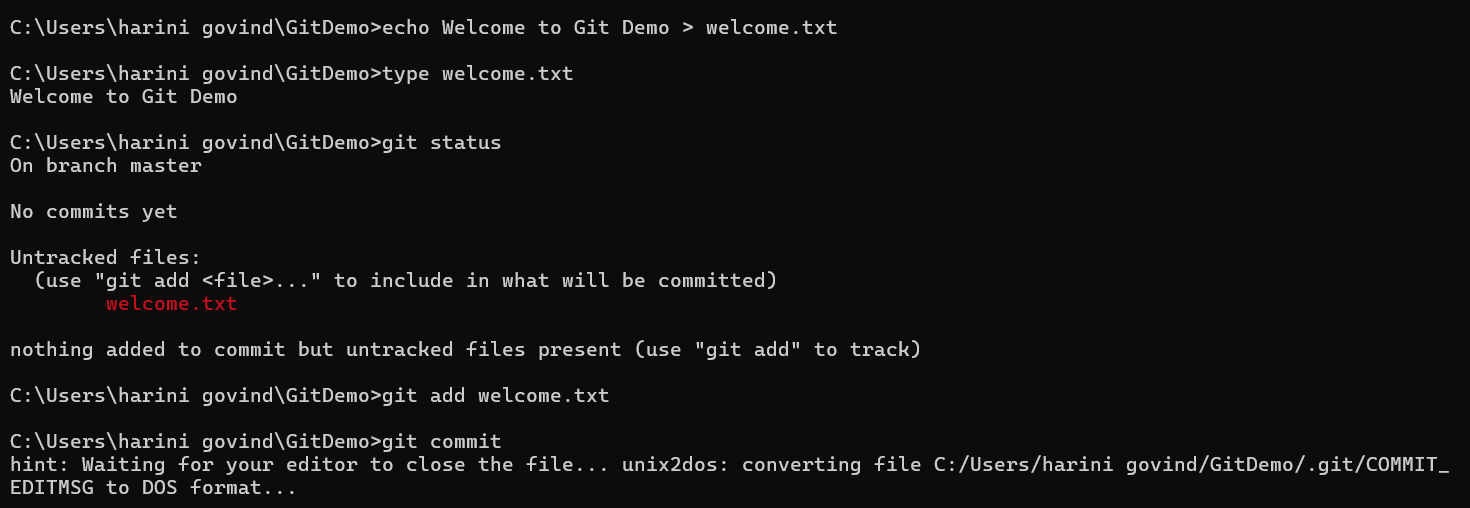
git status

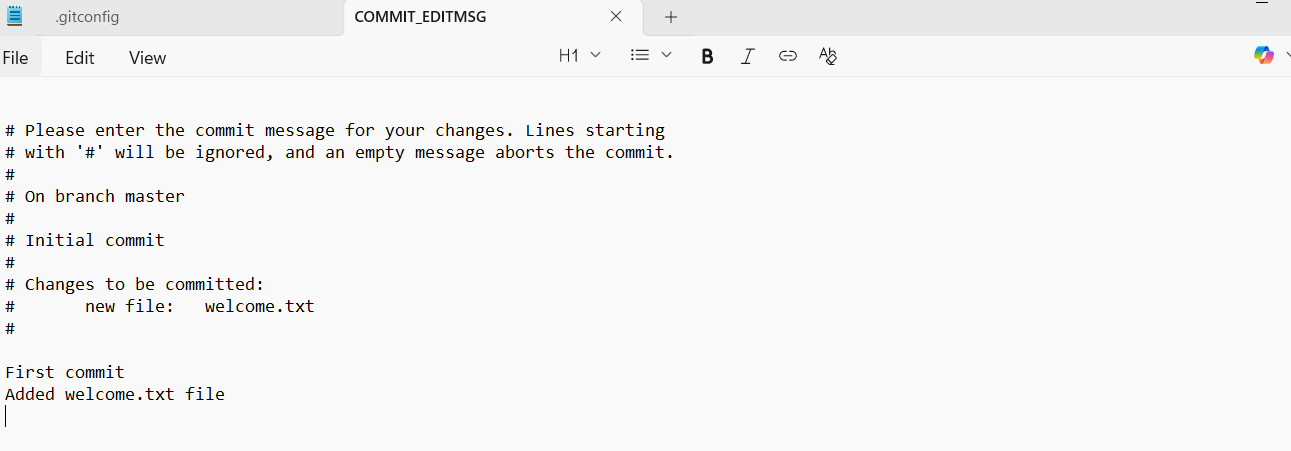
git add welcome.txt

git commit

git status







**Exercise 2:**

**Step 1 – Create Unwanted Files and Folders**

This step creates a .log file and a log folder inside your Git working directory.

echo "This is a log file" > error.log

mkdir log

echo "Log file inside log folder" > log/debug.log

**Step 2 – Create/Edit .gitignore File**  
This step creates a .gitignore file and specifies patterns to ignore .log files and the log folder.

notepad .gitignore

\*.log

log/

**Step 3 – Verify .gitignore is Working**  
This step checks that Git is ignoring the specified files and folders.

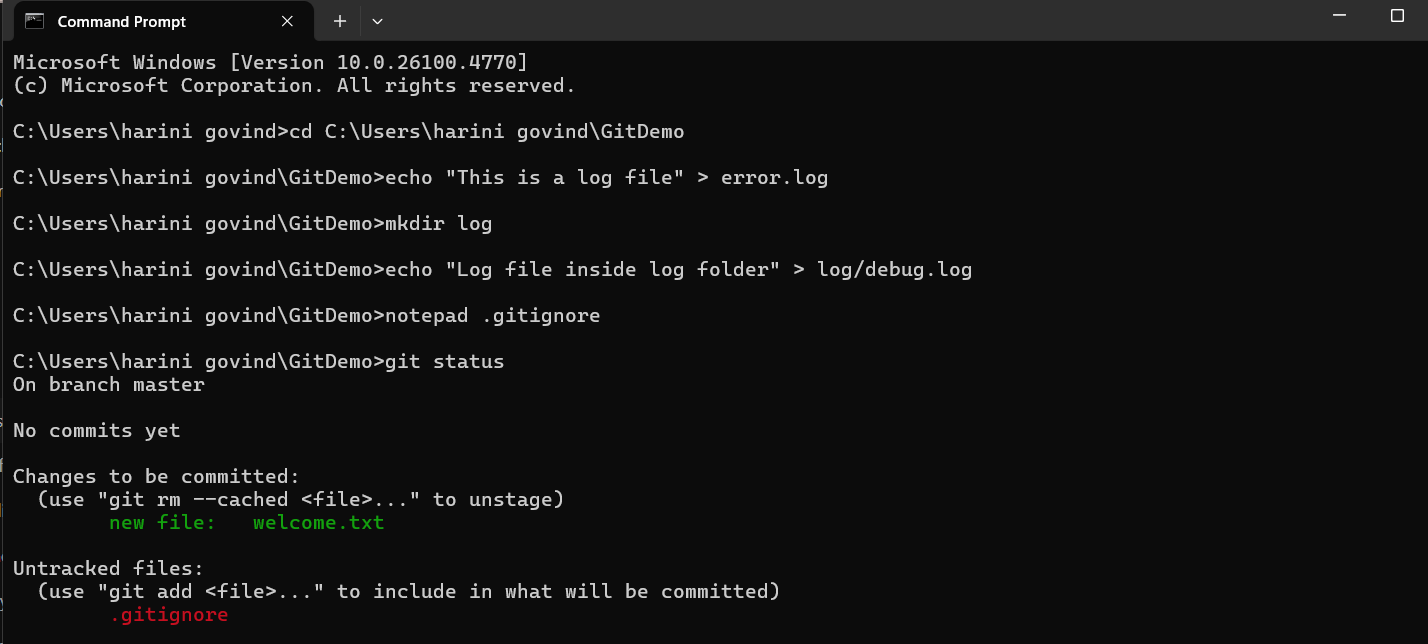
git status

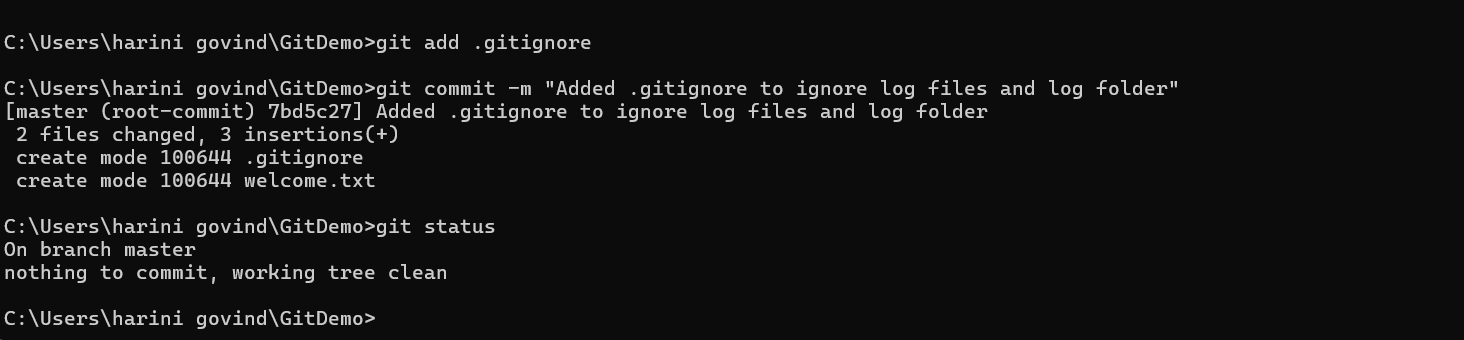
**Step 4 – Commit Changes Without Ignored Files**  
This step commits only the .gitignore file and other tracked files, leaving ignored files out of the commit.

git add .gitignore

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

git status

****

****

**Exercise 3:**

**Branching and Merging**

* Branching in Git allows you to create a separate line of development from the main project.  
  You can work on new features, bug fixes, or experiments without affecting the main codebase.
* Merging is the process of combining changes from one branch into another.  
  After work in a branch is complete, merging integrates it back into the main branch (often called master or main).

**Creating a Branch Request in GitLab**

* In GitLab, a branch request usually means creating a new branch in the remote repository so that other team members can collaborate on it.
* This can be done from:
  + GitLab Web UI → Repository → Branches → New branch.
  + Git Commands:

git checkout -b feature-branch

git push -u origin feature-branch

Creating a Merge Request in GitLab

* A Merge Request (MR) in GitLab is a formal request to merge changes from one branch into another (e.g., feature-branch → master).
* It allows code review, discussion, and approval before merging.
* **Steps in GitLab:**
  1. Push your branch to GitLab.
  2. Go to the repository in GitLab and click Merge Requests → New Merge Request.
  3. Select source branch and target branch.
  4. Add a title, description, and reviewers.
  5. Submit the merge request for review and approval.

**Git commands:**

git checkout -b GitNewBranch

git branch -a

echo "This is a new file in GitNewBranch" > branchfile.txt

git add branchfile.txt

git commit -m "Add branchfile.txt in GitNewBranch"

git status

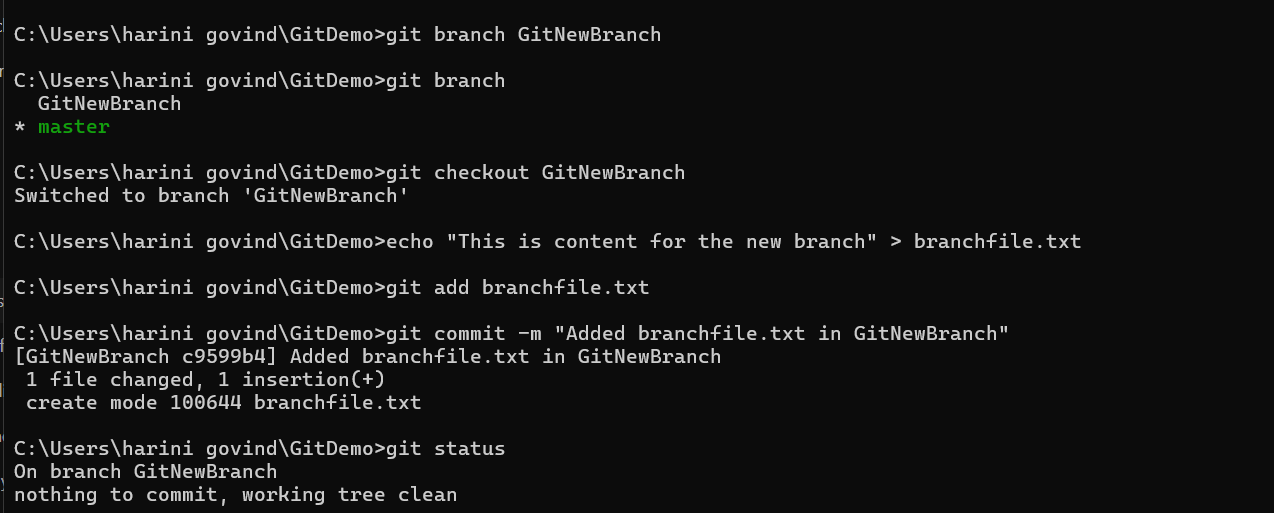
git checkout master

git diff master GitNewBranch

git merge GitNewBranch

git log --oneline --graph –decorate

git branch -d GitNewBranch



**Exercise 4:**

**Merge Conflict**

A merge conflict occurs when **two commits modify the same part of a file** and Git can’t automatically decide which change to keep.  
Example:

* Branch master changes line 5 of hello.xml.
* Branch GitWork also changes line 5 of hello.xml but with different content.

When merging, Git will ask you to resolve it manually.

**Git commands:**

git checkout master

git status

git checkout -b GitWork

echo "<message>Hello from branch</message>" > hello.xml

git add hello.xml

git commit -m "Added hello.xml in GitWork branch"

git checkout master

echo "<message>Hello from master</message>" > hello.xml

git add hello.xml

git commit -m "Added hello.xml in master branch"

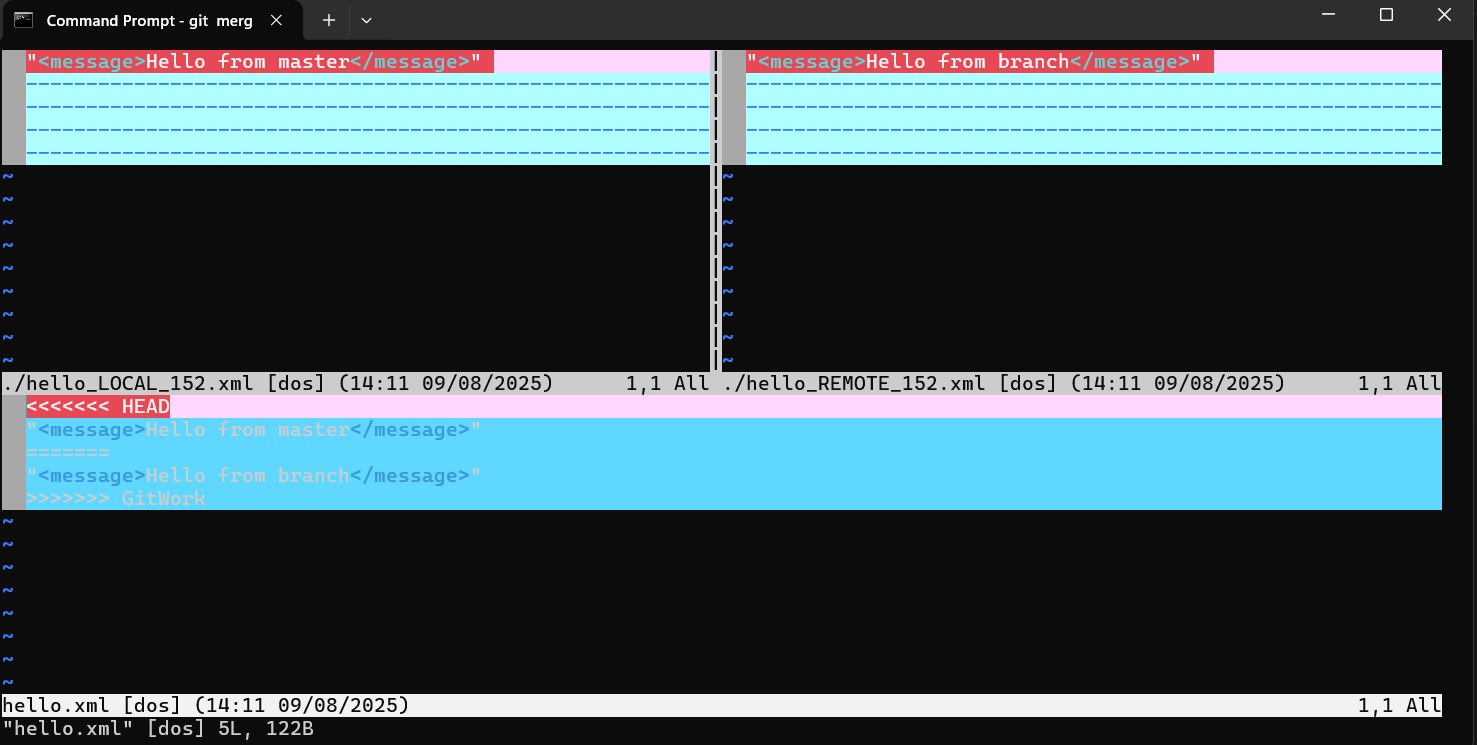
git log --oneline --graph --decorate –all

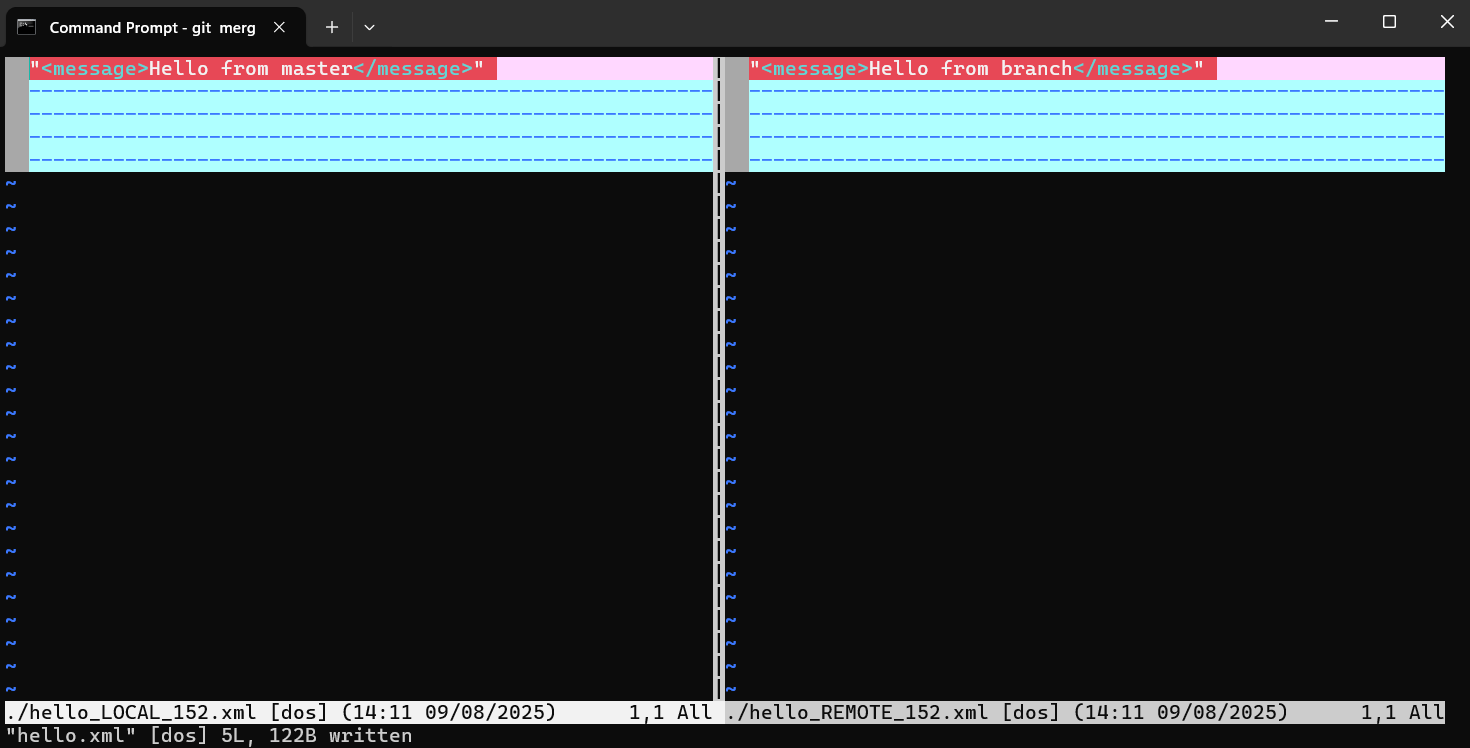
git merge GitWork

git mergetool

git add hello.xml

git commit -m "Resolved merge conflict between master and GitWork"





**Exercise 5:**

In Git, “clean up” means ensuring that your local repository is in a stable and consistent state before sending changes to a remote repository.  
This typically involves:

* Making sure you are on the correct branch (e.g., master or main)
* Committing or stashing uncommitted changes
* Synchronizing with the remote repository to avoid conflicts

Once cleaned up, you “push back” changes to the remote repository so that others can access your updated work.

**Git commands:**

git checkout master

git status

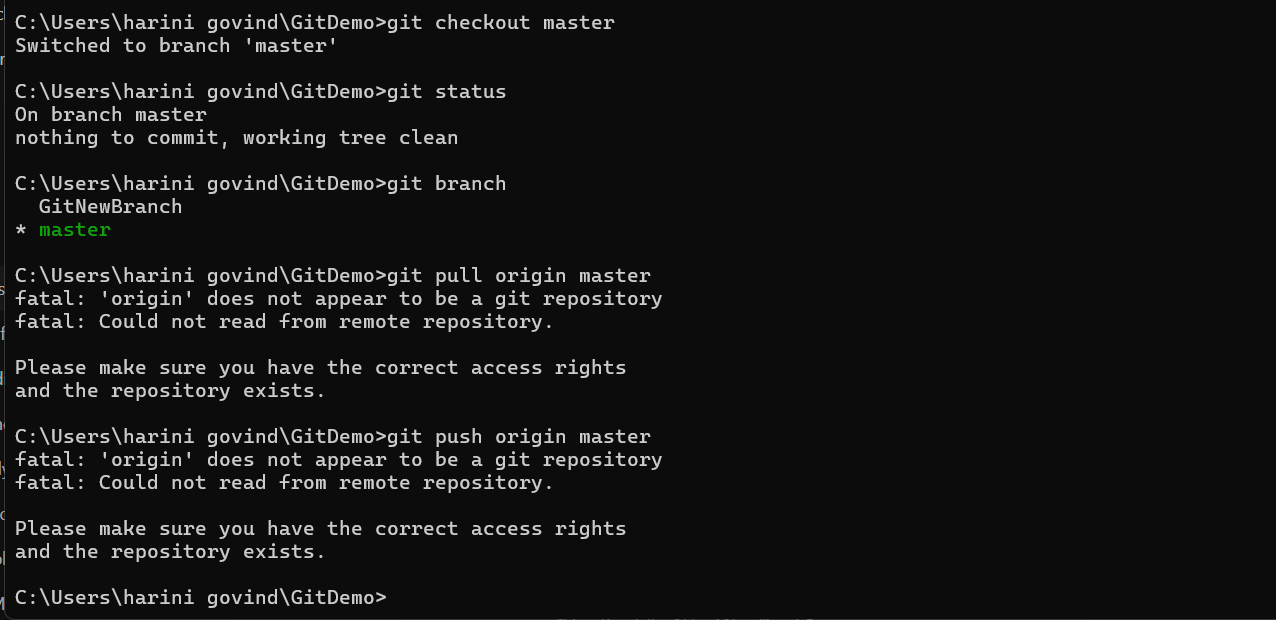
git branch -a

git pull origin master

git push origin master

git checkout GitNewBranch

git push origin GitNewBranch



**NAME: HARINI G  
SUPERSET ID: 6385037**