**Week – 8**

**1.Git Hands-On Lab**

**Objective**

Familiarize yourself with Git commands (git init, git status, git add, git commit, git push, git pull) and basic configuration.  
You will:

* Set up Git configuration
* Set default editor
* Create a file and commit to repository
* Push changes to GitLab

**Step 1: Set Up Git Configuration**

1. Open **Git Bash**.
2. Check Git installation:

git –version

3.Set your name and email:

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

git config --global user.email "your@email.com"

4.Verify settings:

git config --list

**Step 2: Set Default Editor**

1. Set Notepad as default:

git config --global core.editor "notepad"

2. Verify:

git config --global -e

**Step 3: Create Local Repository**

1. Create a project folder:

mkdir GitDemo

cd GitDemo

2.Initialize Git:

git init

3.Check hidden files:

ls -a

**Step 4: Create and Add a File**

1. Create file:

notepad welcome.txt

2.Check file:

ls

cat welcome.txt

3.Stage the file:

git add welcome.txt

4.Commit the file:

git commit -m "Add welcome.txt"

**Step 5: Push to GitLab**

1. On GitLab, create a new project named **GitDemo**.
2. Copy the repository HTTPS link (e.g., https://gitlab.com/username/GitDemo.git).
3. Link your local repo to GitLab:

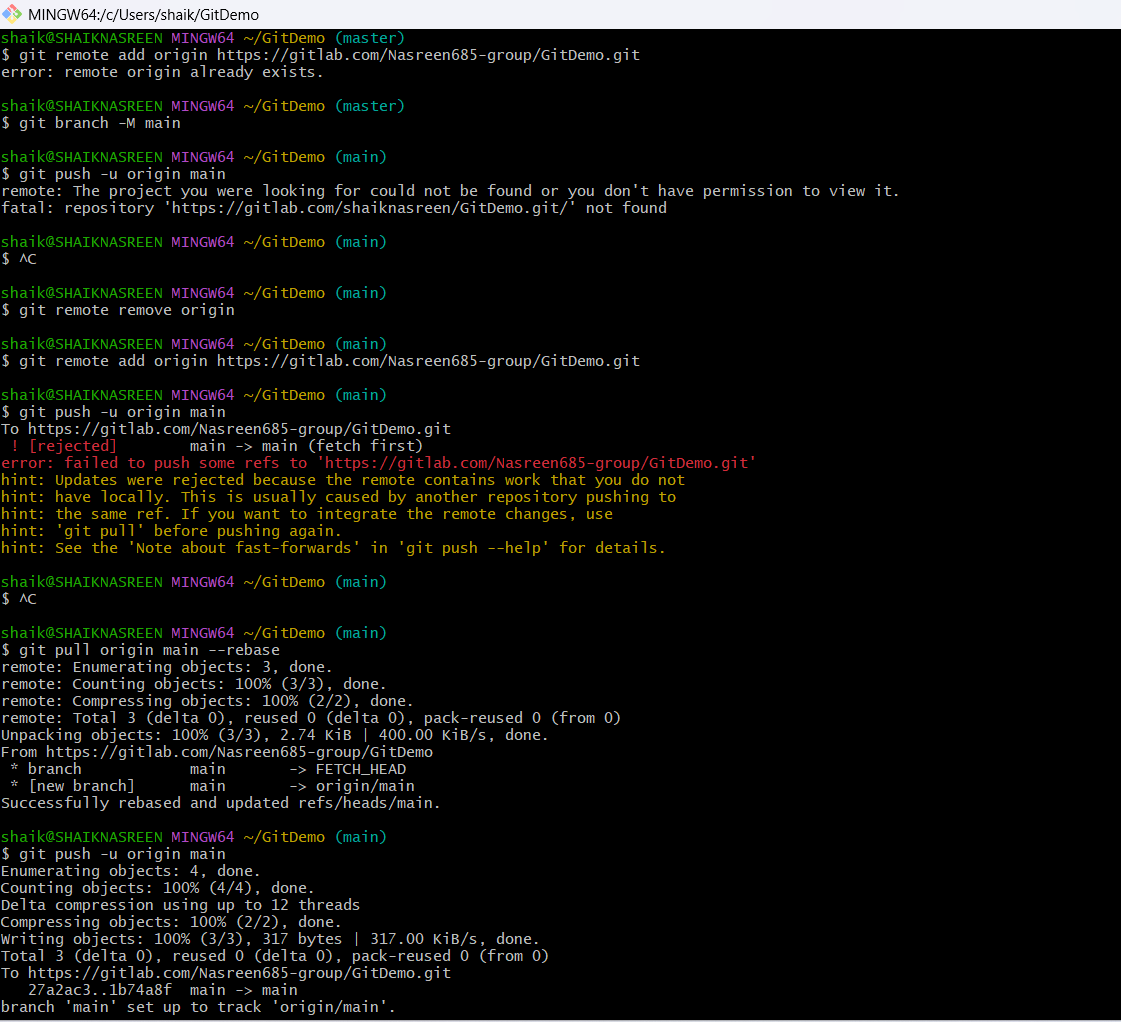
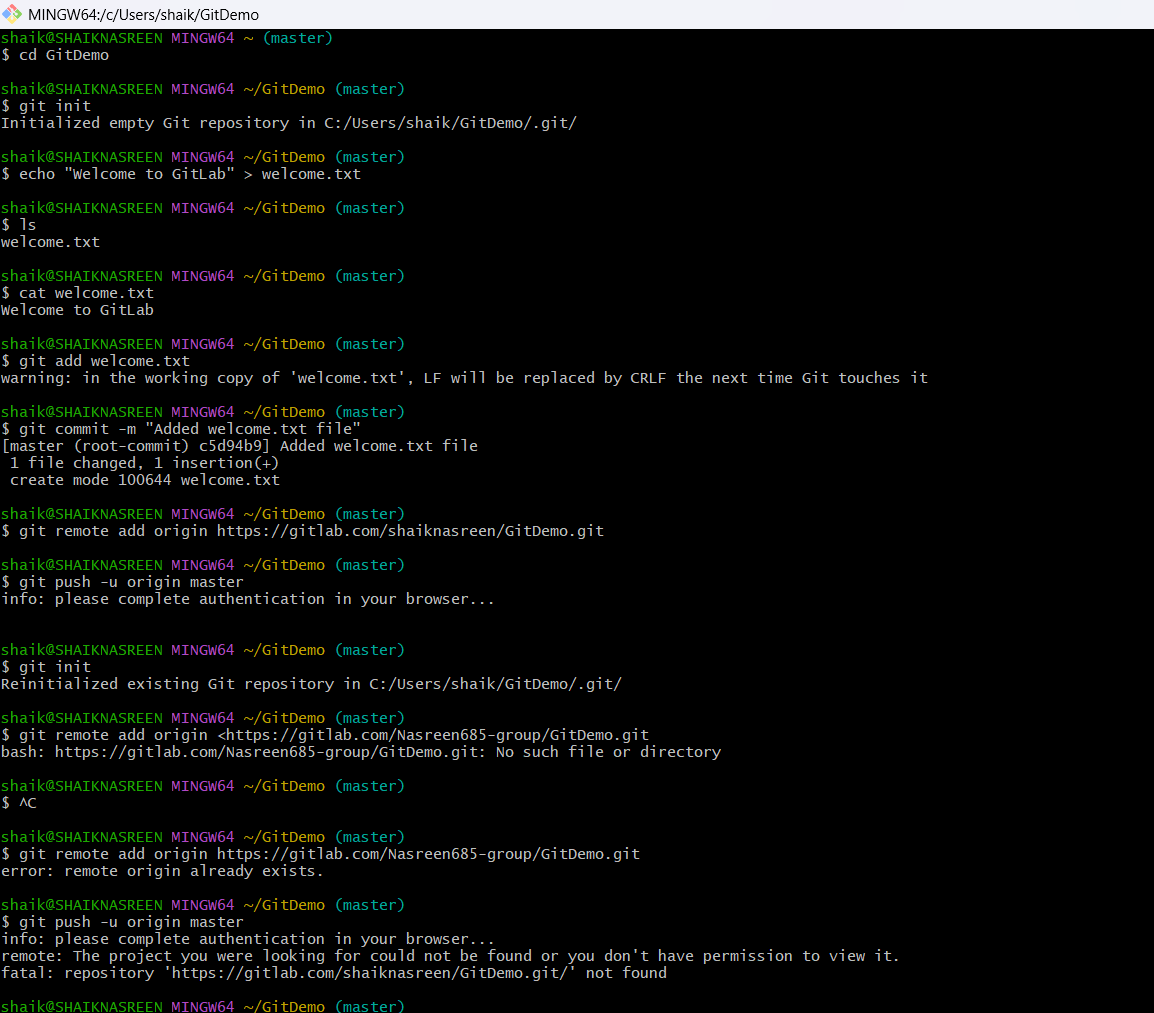
git remote add origin https://gitlab.com/username/GitDemo.git

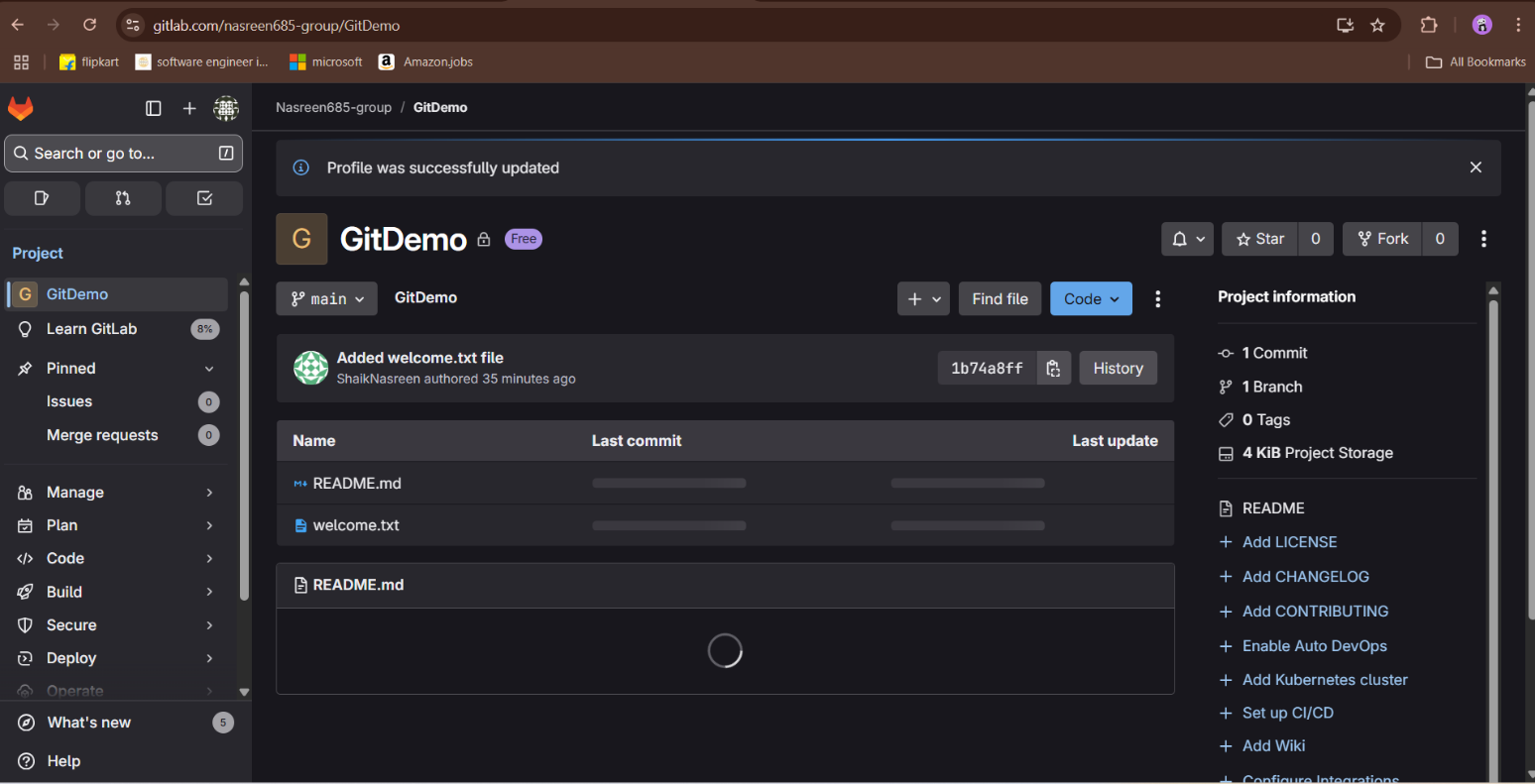
4.Push your local files:

git push -u origin main

**Step 6: Verify on GitLab**

* Go to your GitLab project page.
* In the **Files** tab, you should see welcome.txt and your commit message.





**2.Hands-on Lab: Using .gitignore to Ignore Unwanted Files in Git**

**Step 1 — Open the local repository**

1. Open **Git Bash**.
2. Navigate to your Git working directory.  
   Example:

cd ~/GitDemo

*This changes your directory to the local Git repository GitDemo.*

**Step 2 — Create files to ignore**

1. Create a .log file and a log folder:

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

mkdir log

echo "Log data" > log/data.txt

*These files will later be ignored by Git.*

**Step 3 — Create. gitignore file**

1. Open. gitignore in Notepad:

notepad. gitignore

1. Add the following lines to ignore .log files and log folder:

\*.log

log/

1. Save and close Notepad.

**Step 4 — Check status in Git**

Run:

git status

You should see. gitignore listed as untracked, but **test.log** and log/ are not listed because they are ignored.

**Step 5 — Add and commit. gitignore**

bash

Copy code

git add .gitignore

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

**Step 6 — Push changes to GitLab**

git push -u origin main

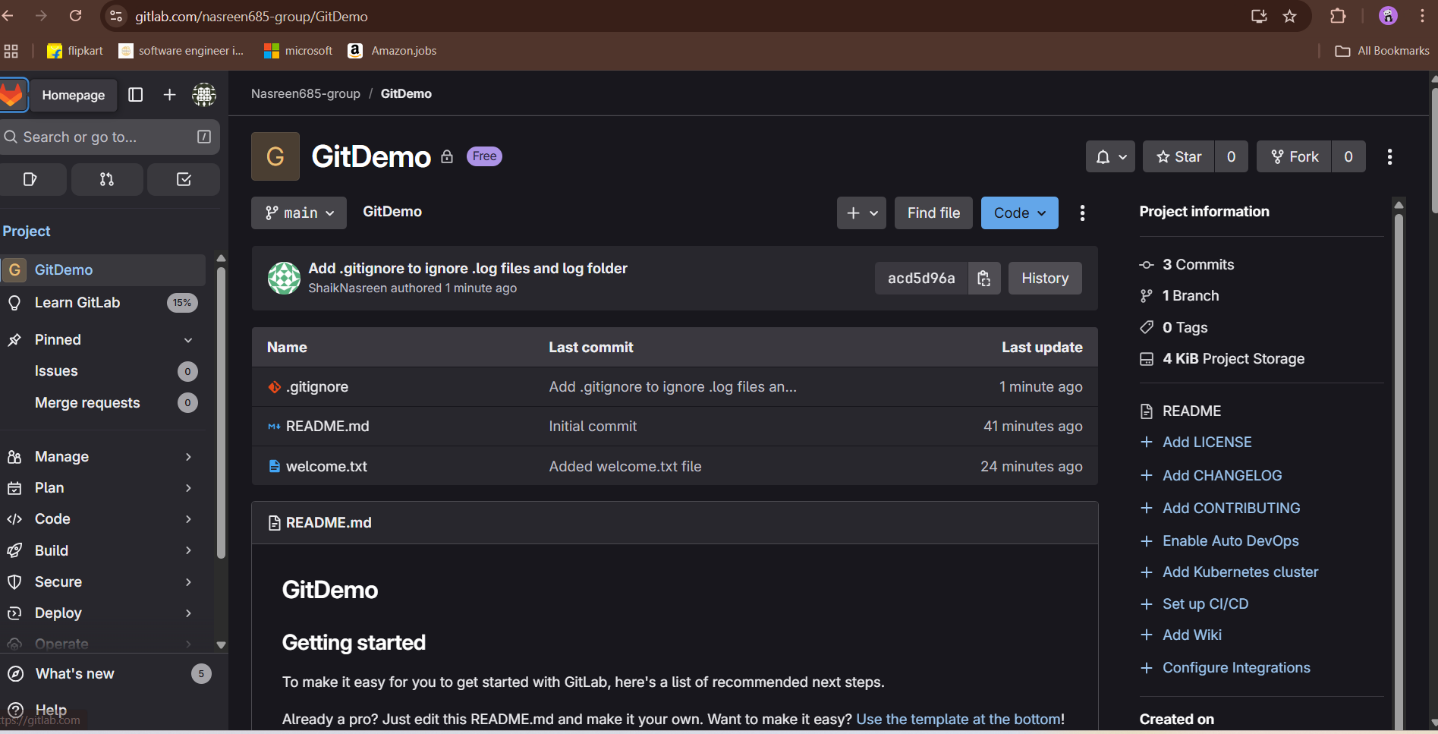
*Your .gitignore is now pushed to the remote repository.*

**Step 7 — Verify in GitLab**

1. Open your GitLab project:  
   https://gitlab.com/Nasreen685-group/GitDemo.git
2. You should see .gitignore listed in the repository.

**Explanation**

* **.gitignore** tells Git which files or directories to skip when tracking changes.
* The line \*.log ignores all .log files.
* The line log/ ignores any folder named log.



**3.Hands-on: Branching and Merging in GitLab**

Objectives

* Explain branching and merging in Git.
* Explain how to create a branch request in GitLab.
* Explain how to create a merge request in GitLab.

Branching Steps

1. Create a new branch

git branch GitNewBranch

2. List all local and remote branches

git branch -a

* The \* symbol indicates the current active branch.

3. Switch to the new branch

git checkout GitNewBranch

4. Add some files with content

echo "This is new content for GitNewBranch" > newfile.txt

git add newfile.txt

5. Commit the changes to the branch

git commit -m "Added newfile.txt in GitNewBranch"

6. Check branch status

git status

Merging Steps

1. Switch back to master branch

git checkout master

2. View differences between master and branch (CLI)

git diff GitNewBranch

3. View differences using P4Merge (GUI)

git mergetool

4. Merge the branch into master

git merge GitNewBranch

5. View merge history

git log --oneline --graph --decorate

6. Delete the merged branch

git branch -d GitNewBranch

7. Check status after merge

git status

Creating a Branch Request in GitLab

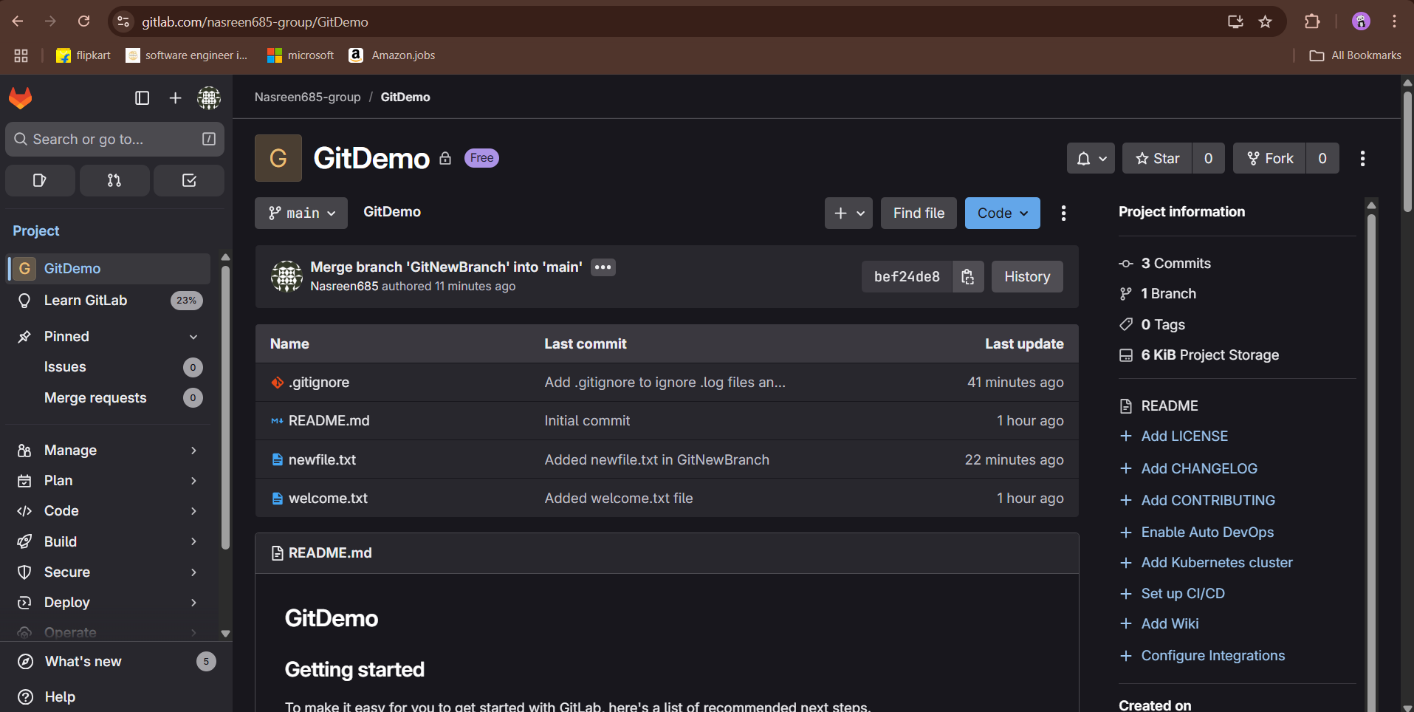
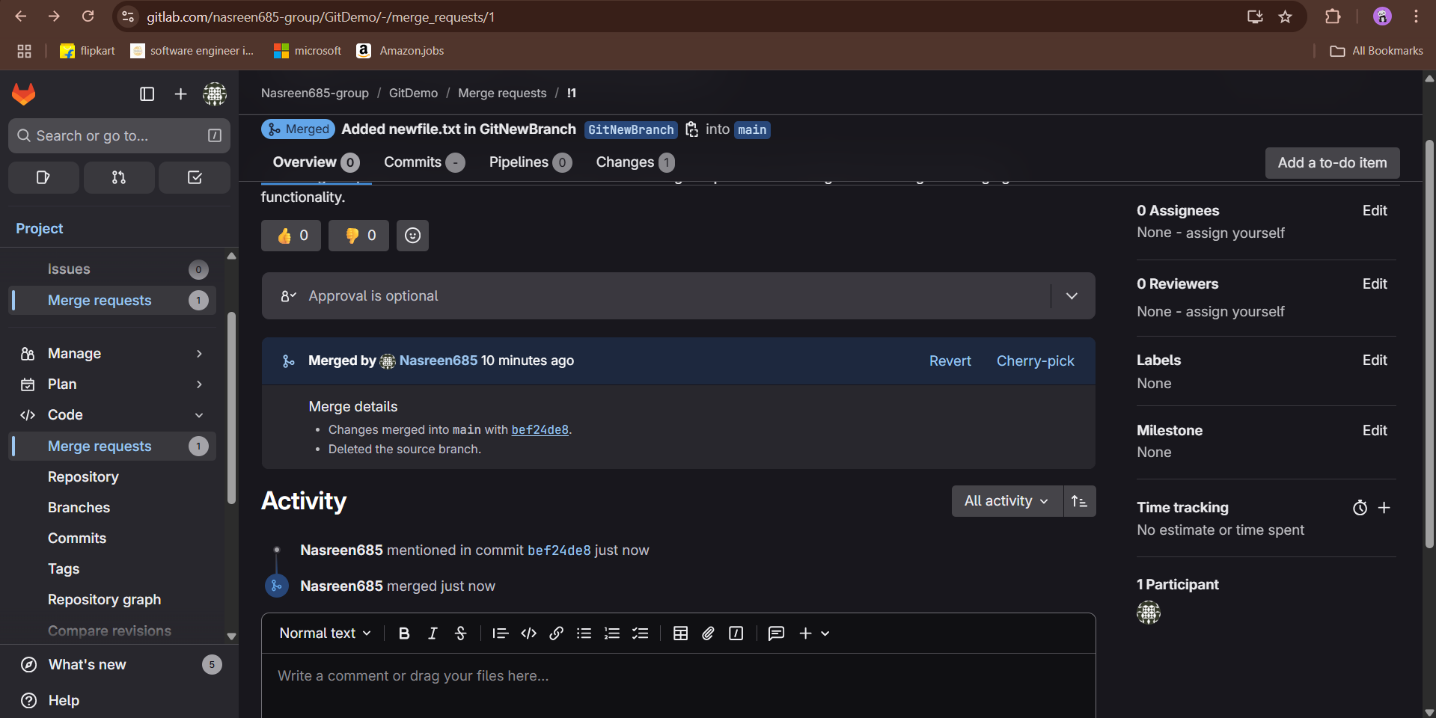
1. Push the branch to GitLab:

git push origin GitNewBranch

1. Go to your project in GitLab.
2. Click on Branches under Repository.
3. Find GitNewBranch and click New Merge Request.

Creating a Merge Request in GitLab

1. In GitLab, click Merge Requests > New Merge Request.
2. Select source branch as GitNewBranch and target branch as master.
3. Review changes and click Submit.
4. Once approved, click Merge to merge changes into master.



**4.Git Merge Conflict Resolution Lab**

**Objective:**  
Explain how to resolve conflicts during a Git merge when multiple users update the same files on different branches.

**Steps and Commands**

**1. Verify current branch status**

git status

**2. Create and switch to a new branch GitWork**

git checkout -b GitWork

**3. Add and commit hello.xml file on GitWork**

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

git add hello.xml

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

**4. Switch back to main branch**

git checkout main

**5. Add and commit different hello.xml content on main**

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

git add hello.xml

git commit -m "Add hello.xml in main branch"

**6. Observe commit history**

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

**7. Check differences between branches**

git diff main GitWork

**8. Merge GitWork into main (causes conflict)**

git merge GitWork

**9. Resolve conflict manually or with merge tool**

* Open hello.xml
* Edit the file to combine changes, removing conflict markers:

<message>Hello from both branches</message>

<note>This is branch content</note>

* Save the file.

**10. Stage and commit resolved file**

git add hello.xml

git commit -m "Resolved merge conflict in hello.xml"

**11. Add backup files to .gitignore**

echo "\*.orig" >> .gitignore

git add .gitignore

git commit -m "Add merge backup files to .gitignore"

**12. Delete merged branch GitWork**

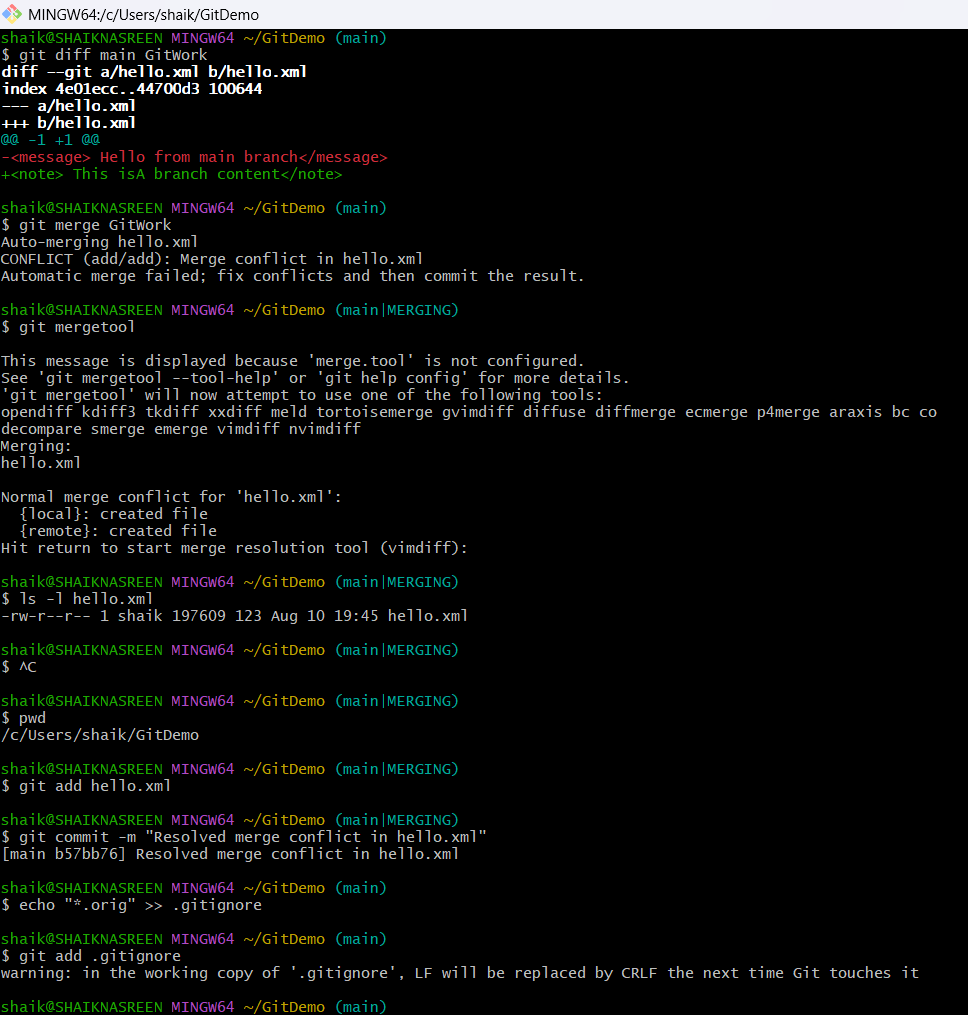
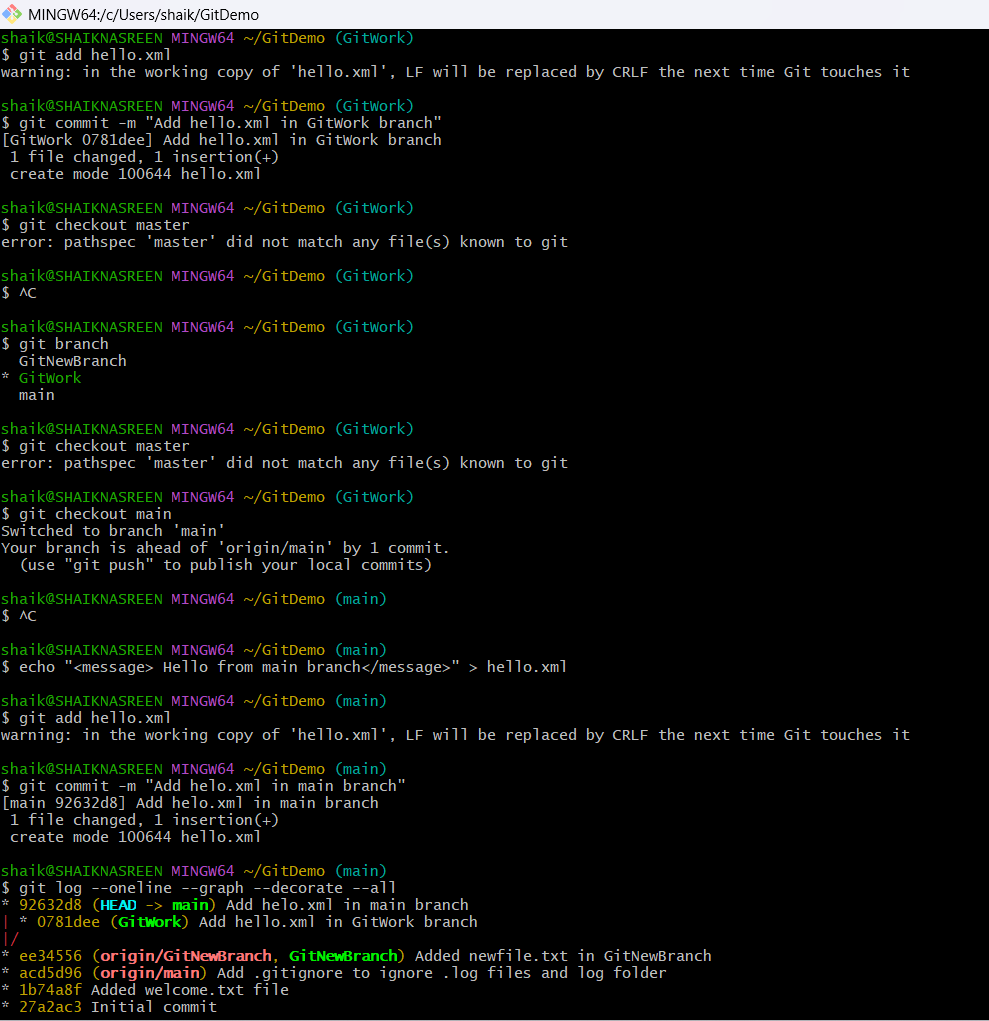
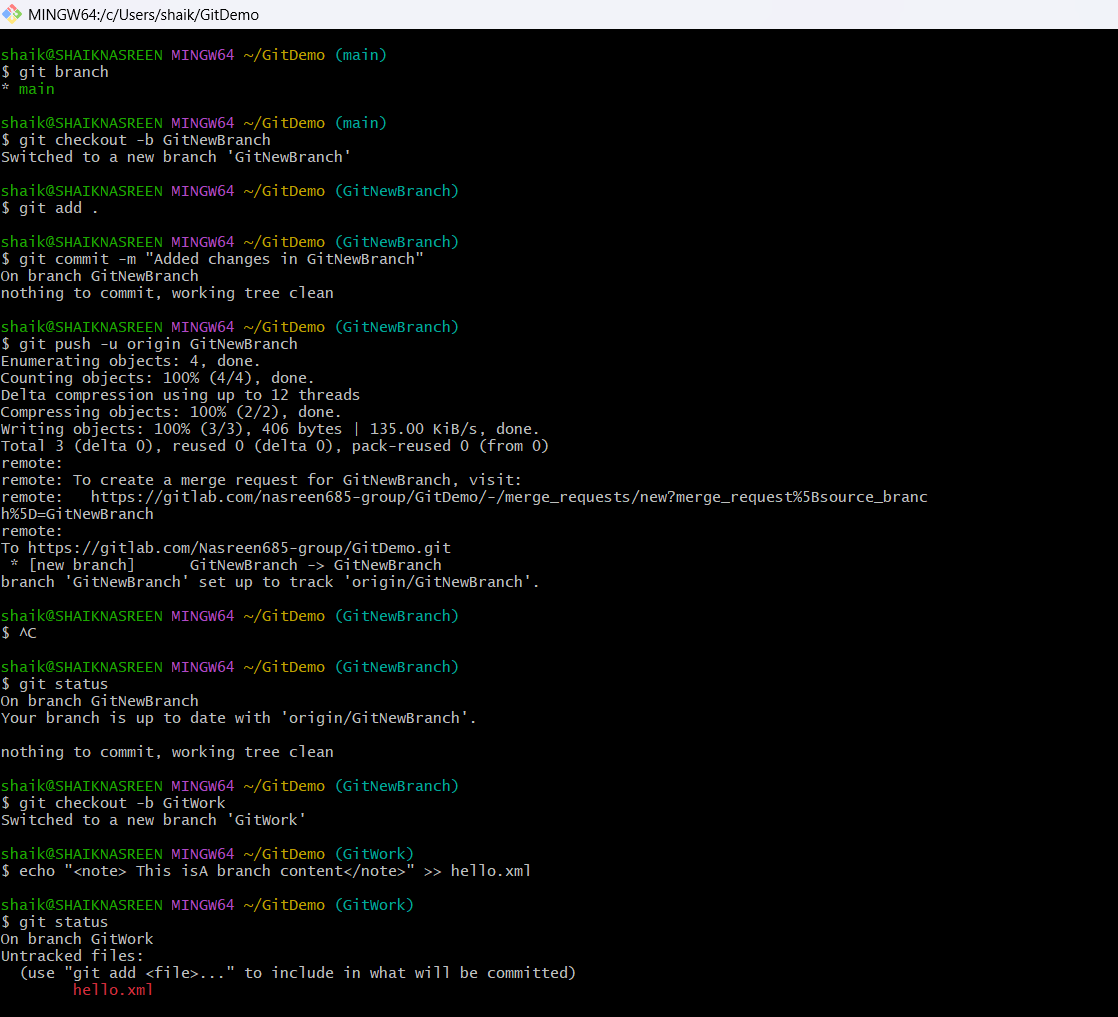
git branch -d GitWork

**13. Final commit history check**

git log --oneline --graph --decorate

**Conclusion**

* The conflict in hello.xml was successfully resolved.
* Backup files were ignored via .gitignore.
* The feature branch GitWork was deleted after merging.
* The project is now clean and up to date on main.



**5.Git Clean-Up and Push Back Lab**

**Objective:**  
Explain how to clean up your local branch and push changes back to remote Git repository.

**Steps and Commands**

**1. Verify if main branch is in a clean state**

git status

* Confirmed branch is ahead of remote by 5 commits.
* Noted untracked backup files present.

**2. Add merge backup files to .gitignore (optional but recommended)**

echo "hello\_BACKUP\_910.xml" >> .gitignore

echo "hello\_BASE\_910.xml" >> .gitignore

echo "hello\_LOCAL\_910.xml" >> .gitignore

echo "hello\_REMOTE\_910.xml" >> .gitignore

git add .gitignore

git commit -m "Ignore merge backup files"

**3. Pull latest changes from remote and rebase local commits**

git pull origin main --rebase

* Handled merge commit message in Vim editor by saving and exiting.

**4. Push local commits to remote repository**

git push origin main

* Successfully pushed 5 commits to remote without errors.

**Result**

* Local and remote branches are synced.
* Backup files are ignored by Git.
* Remote repository on GitLab reflects all latest commits.
* 