Assignment – 01:

**Step 1 – Setup Git Configuration**

1. **Check if Git is installed**

bash

git --version

1. **Configure your Git username and email**

bash

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

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

**Step 2 – Make Notepad++ the Default Editor**

1. **Check if Notepad++ is accessible from Git Bash**

bash

notepad++

If it says *command not found*, add Notepad++ path to your system environment variables:

* **Windows**:  
  Control Panel → System → Advanced System Settings → Environment Variables  
  Find **Path** → Edit → Add the path to notepad++.exe (usually "C:\Program Files\Notepad++")

1. **Set Notepad++ as Git’s default editor**

bash

git config --global core.editor "notepad++ -multiInst -nosession"

1. **Verify**

bash

git config --global -e

**Step 3 – Create and Commit a File**

1. **Create a project folder**

bash

mkdir GitDemo

cd GitDemo

1. **Initialize Git repository**

Bash

git init

1. **Create a file and add content**

bash

echo "Welcome to Git" > welcome.txt

1. **Check file**

bash

ls

cat welcome.txt

1. **Check status**

bash

git status

1. **Stage the file**

bash

git add welcome.txt

1. **Commit the file**

bash

git commit -m "Add welcome.txt with initial content"

**Step 4 – Push to Remote Repository**

1. **Create a remote repo** on **GitLab** named GitDemo.
2. **Link it to your local repo**

bash

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

1. **Pull latest changes (optional if repo is empty)**

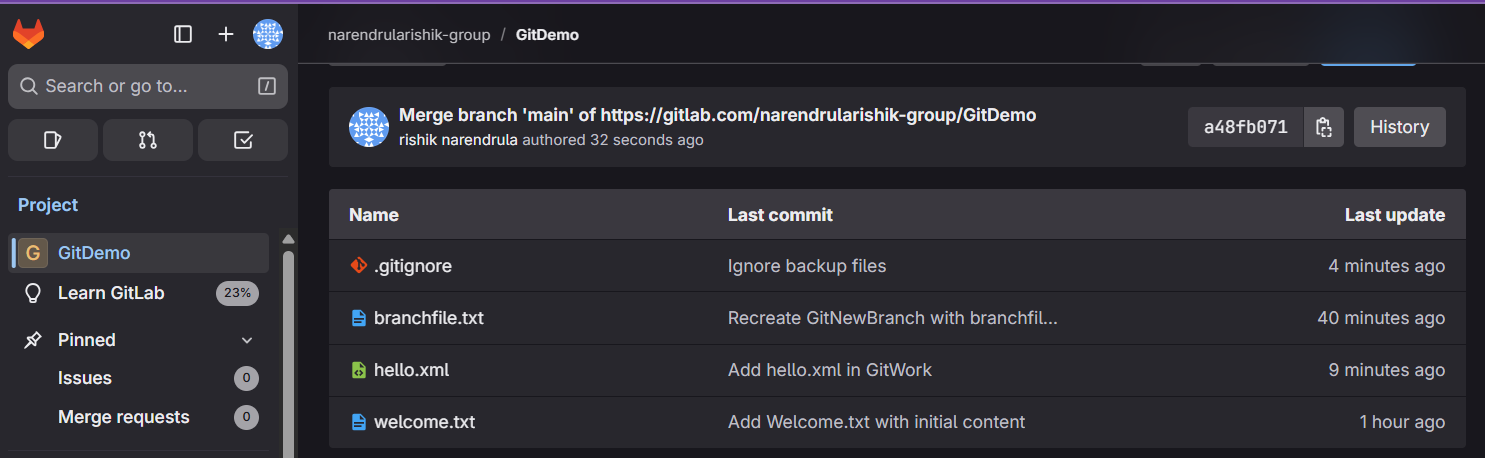
bash

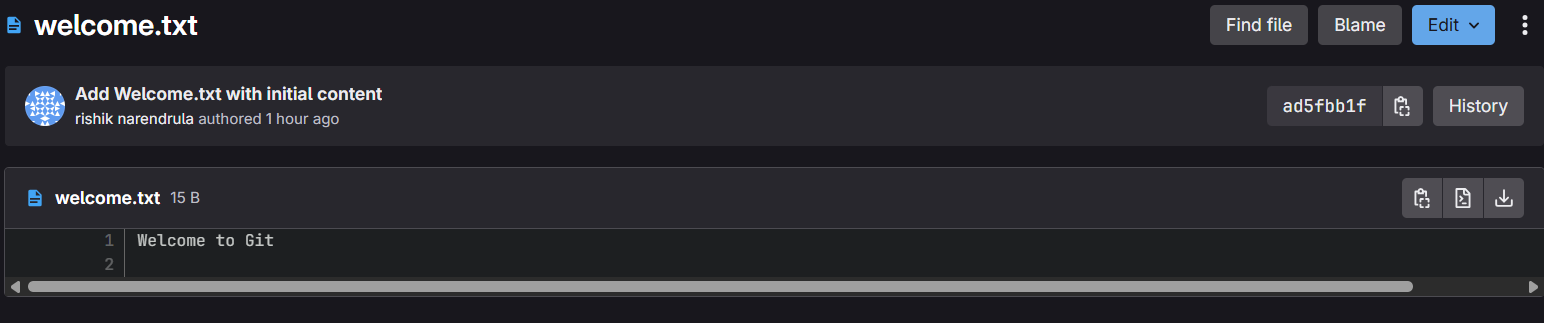
git pull origin master

1. **Push your code**

bash

git push origin master





Assignment – 02:

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

In your Git Bash, inside your project folder (e.g., GitDemo):

Bash

# create a .log file

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

# create a log folder and a file inside it

mkdir log

echo "This is inside log folder" > log/debug.txt

1. **Create a .gitignore file**

bash

touch .gitignore

1. **Add ignore rules**

Open .gitignore in Notepad++ (since you already set it as Git editor):

bash

notepad++ .gitignore

Add these lines:

bash

# Ignore all .log files

\*.log

# Ignore the log folder

log/

Save and close.

1. **Verify that Git is ignoring them**

Run:

bash

git status

**You should not see test.log or anything inside log/ listed under “Untracked files.”  
If they were already tracked before you added them to .gitignore, you must first untrack them:**

bash

git rm --cached test.log

git rm -r --cached log

1. **Commit the .gitignore**

bash

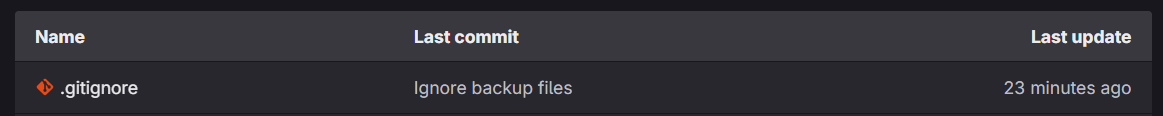
git add .gitignore

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

1. **Push to remote**

bash

git push origin main



Assignment – 03:

**Step 1 — Create a new branch**

bash

git branch GitNewBranch

**Step 2 — List all branches**

bash

git branch -a

* The \* shows your **current branch**.

**Step 3 — Switch to the new branch**

bash

git checkout GitNewBranch

**Step 4 — Add files in the new branch**

bash

echo "This is a new branch file" > branchfile.txt

git add branchfile.txt

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

**Step 5 — Check status**

bash

git status

**2. Merging (Git Bash)**

**Step 1 — Switch back to master/main**

bash

git checkout main

**Step 2 — See differences between branches**

bash

git diff main GitNewBranch

*(Shows text-based differences in terminal.)*

**Step 3 — See visual differences with P4Merge**

If you have **P4Merge** installed and set as your diff tool:

bash

git difftool main GitNewBranch

If not configured yet:

bash

git config --global diff.tool p4merge

git config --global difftool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"

**Step 4 — Merge the branch into main**

bash

git merge GitNewBranch

**Step 5 — View merge history**

bash

git log --oneline --graph --decorate

**Step 6 — Delete the branch**

bash

git branch -d GitNewBranch

1. **Creating a Merge Request in GitLab (UI)**

If you push the branch to GitLab, you can use GitLab’s Merge Request feature instead of merging locally.

1. **Push the branch to GitLab**:

bash

git push origin GitNewBranch

1. **In GitLab**:
   * Go to your repository → **Repository → Branches**.
   * You’ll see GitNewBranch listed with a **Merge Request** button.
   * Click **Merge Request**.
2. **Fill in the merge request form**:
   * **Source branch**: GitNewBranch
   * **Target branch**: main
   * Add a title and description of your changes.
   * Click **Create merge request**.
3. Review changes → Click **Merge**.
4. After merge, you can delete the branch from GitLab using the **Delete source branch** checkbox.





Assignment – 04:

* 1. **Verify master is clean**

Make sure there are no pending changes:

bash

git checkout main

git status

If you see “nothing to commit, working tree clean,” you’re good.

* 1. **Create branch “GitWork” and add hello.xml**

bash

git checkout -b GitWork

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

git add hello.xml

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

* 1. **Switch back to master**

bash

git checkout main

* 1. **Add a conflicting hello.xml in master**

bash

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

git add hello.xml

git commit -m "Add hello.xml in master"

* 1. **View commit history**

bash

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

You’ll see two separate commits with hello.xml — one in GitWork, one in main.

* 1. **View differences**

Text-based:

bash

git diff main GitWork

Visual (if P4Merge is installed and set):

bash

git config --global diff.tool p4merge

git config --global difftool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"

git difftool main GitWork

* 1. **Merge GitWork into master**

bash

git merge GitWork

This will produce a **merge conflict** in hello.xml.

* 1. **Resolve the conflict (3-way merge)**

If P4Merge is your merge tool:

bash

git config --global merge.tool p4merge

git config --global mergetool.p4merge.path "C:/Program Files/Perforce/p4merge.exe"

git mergetool

P4Merge will show **3 panels**:

* Left: master version
* Right: GitWork branch version
* Middle/Bottom: merged result (edit this to resolve)

For the lab, you can combine them like:

xml

<message>Hello from master and branch</message>

Save and close P4Merge.

* 1. **Mark conflict as resolved and commit**

bash

git add hello.xml

git commit -m "Merge GitWork into master and resolve conflict"

* 1. **Add backup file to .gitignore**

When using mergetool, P4Merge might create backup files like hello.xml.orig.  
To ignore them:

bash

echo "\*.orig" >> .gitignore

git add .gitignore

git commit -m "Ignore backup files"

* 1. **Delete branch after merge**

bash

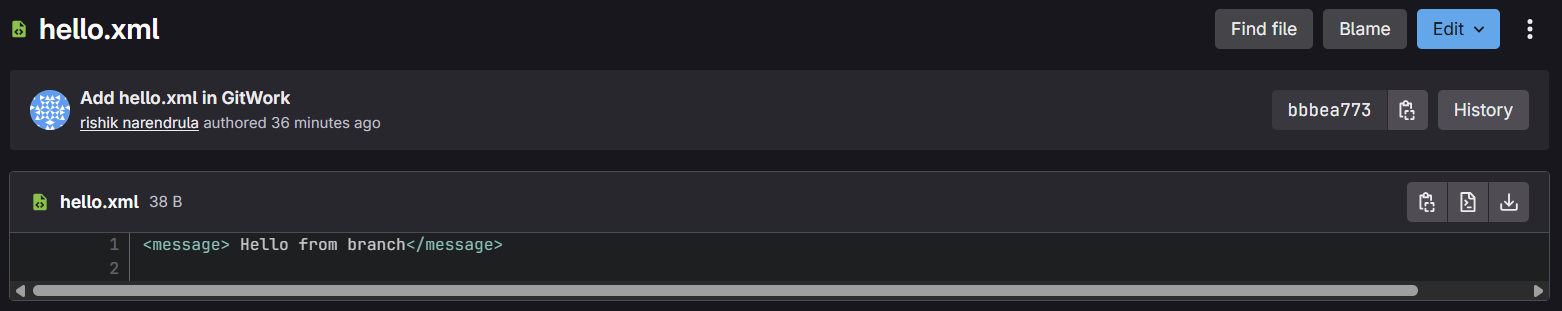
git branch -d GitWork

* 1. **View final commit history**

bash

git log --oneline --graph --decorate





Assignment – 05:

1. **Verify if master is in clean state**

bash

git checkout main # or master, depending on your branch name

git status

If it says:

pgsql

nothing to commit, working tree clean

then you’re good.  
If there are changes, either commit them or discard them before continuing.

1. **List out all available branches**

bash

git branch -a

* Local branches will be listed normally.
* Remote branches will be listed as remotes/origin/<branch-name>.

1. **Pull the remote repository into master**

bash

git pull origin main

This will fetch and merge the latest remote changes into your local branch.  
If a merge conflict appears, resolve it, then:

bash

git add .

git commit -m "Resolve merge conflict from remote"

1. **Push any pending changes to remote**

If you already committed changes locally and want to push them:

bash

git push origin main

This will upload your commits to GitLab.

1. **Verify changes on GitLab**

* Go to your GitLab project page.
* Open the **main** branch in the repository browser.
* You should see your files and the latest commit message you pushed.

