**GIT**

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

1. **Check if Git is installed**

git --version

If it shows Git with its version information, Git is installed properly.

1. **Configure user details**

git config --global user.name "Your Name

git config --global user.email [youremail@example.com](mailto:youremail@example.com)  
**A screen shot of a computer program

AI-generated content may be incorrect.**

1. **Verify configuration**

git config --list

1. **Create a new repository in GitLab**

* Signup/Login to GitLab.
* Create a project named **GitDemo**.  
  **A screenshot of a computer program

  AI-generated content may be incorrect.**

**Step 2: Integrate Notepad++ with Git as Default Editor**

1. **Check if Notepad++ is recognized in Git Bash**

notepad++

If not recognized, add the path of notepad++.exe to the **Environment Variables**:

* Control Panel → System → Advanced System Settings → Environment Variables.
* Edit **Path** under User variables and add the location of notepad++.exe.

1. **Restart Git Bash** and verify:

notepad++  
**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Create an alias for Notepad++**

alias np='notepad++'

*(You can also add this alias to your .bash\_profile so it loads every time.)*

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

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

1. **Verify default editor**

git config -e –global  
**A screenshot of a computer

AI-generated content may be incorrect.**

**Step 3: Add a File to Source Code Repository**

1. **Create a new Git project folder**

mkdir GitDemo

cd GitDemo

1. **Initialize the repository**

git init

1. **Verify hidden Git files**

ls -a

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

echo "Welcome to Git Demo" > welcome.txt

1. **Verify file creation**

ls

1. **View file content**

cat welcome.txt

1. **Check Git status**

git status

1. **Stage the file for commit**

git add welcome.txt

1. **Commit with multi-line comment (opens Notepad++)**

git commit

*(Add your multi-line message in Notepad++ and save.)*

1. **Check repository status**

git status

**Step 4: Push to GitLab Remote Repository**

1. **Create a remote repository on GitLab** named **GitDemo**.
2. **Link local repository to remote**

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

1. **Pull from remote (optional, to sync)**

git pull origin master

1. **Push local repository to remote**

git push origin master  
**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise 2:  
Step 1 — Create a log file and log folder in the working directory**

mkdir log

touch log/test.log

touch error.log

**Step 2 — Create or open .gitignore file**

touch .gitignore

nano .gitignore  
**A screen shot of a computer

AI-generated content may be incorrect.**

**Step 3 — Add rules to ignore .log files and the log folder  
Inside .gitignore, add:**

\*.log

log/

Save and exit.

**Step 4 — Verify .gitignore changes**

git status

Expected output should show .log files and log/ folder not being tracked.

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

git add .gitignore

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

**Part 3 — Verify Status in All Areas**

1. **Working directory  
   Run:**

git status

You should see no .log files listed under untracked files.

1. **Local repository  
   Check commit history:**

git log --oneline

1. **Remote repository  
   Push changes:**

git push origin master

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise 3:**

**Branching**

1. **Create a new branch “GitNewBranch”**

git branch GitNewBranch

1. **List all the local and remote branches**

git branch -a

The branch with \* is your current branch.

1. **Switch to the newly created branch**

git checkout GitNewBranch

1. **Add some files to the branch with contents**

echo "Sample content" > sample.txt

git add sample.txt

1. **Commit the changes to the branch**

git commit -m "Added sample.txt to GitNewBranch"

1. **Check the status**

git status  
A screenshot of a computer

AI-generated content may be incorrect.

**Merging**

1. **Switch to the master branch**

git checkout master

1. **List all the differences between master and GitNewBranch**

git diff master GitNewBranch

1. **List visual differences using P4Merge** (if installed)

git difftool master GitNewBranch

1. **Merge the source branch into master**

git merge GitNewBranch

1. **Observe the logging after merging**

git log --oneline --graph --decorate

1. **Delete the branch after merging**

git branch -d GitNewBranch

1. **Check the status**

git status

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 4:**

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

git status

1. **Create a branch GitWork**

git checkout -b GitWork

1. **Add a file hello.xml**

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

git add hello.xml

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

1. **Update the content of hello.xml and observe the status**

echo "<message>Updated content in GitWork</message>" > hello.xml

git status

1. **Commit the changes to reflect in the branch**

git add hello.xml

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

1. **Switch to master**

git checkout master

1. **Add a file hello.xml to the master with different content**

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

git add hello.xml

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

1. **Observe the log**

git log --oneline --graph --decorate –all  
A screenshot of a computer

AI-generated content may be incorrect.

1. **Check the differences with Git diff tool**

git diff master GitWork

1. **Use P4Merge tool for better visualization**

git difftool master GitWork

1. **Merge the branch into master**

git merge GitWork

1. **Observe the git merge markup** (conflicts will be shown in the file)
2. **Use 3-way merge tool to resolve the conflict**

git mergetool

1. **Commit the changes to master after resolving the conflict**

git add hello.xml

git commit -m "Merge GitWork into master with conflict resolution"  
A screenshot of a computer

AI-generated content may be incorrect.

1. **Observe the git status and add backup file to .gitignore**

git status

echo "\*.bak" >> .gitignore

1. **Commit the .gitignore changes**

git add .gitignore

git commit -m "Update .gitignore to ignore backup files"

1. **List out all the available branches**

git branch  
A screenshot of a computer

AI-generated content may be incorrect.

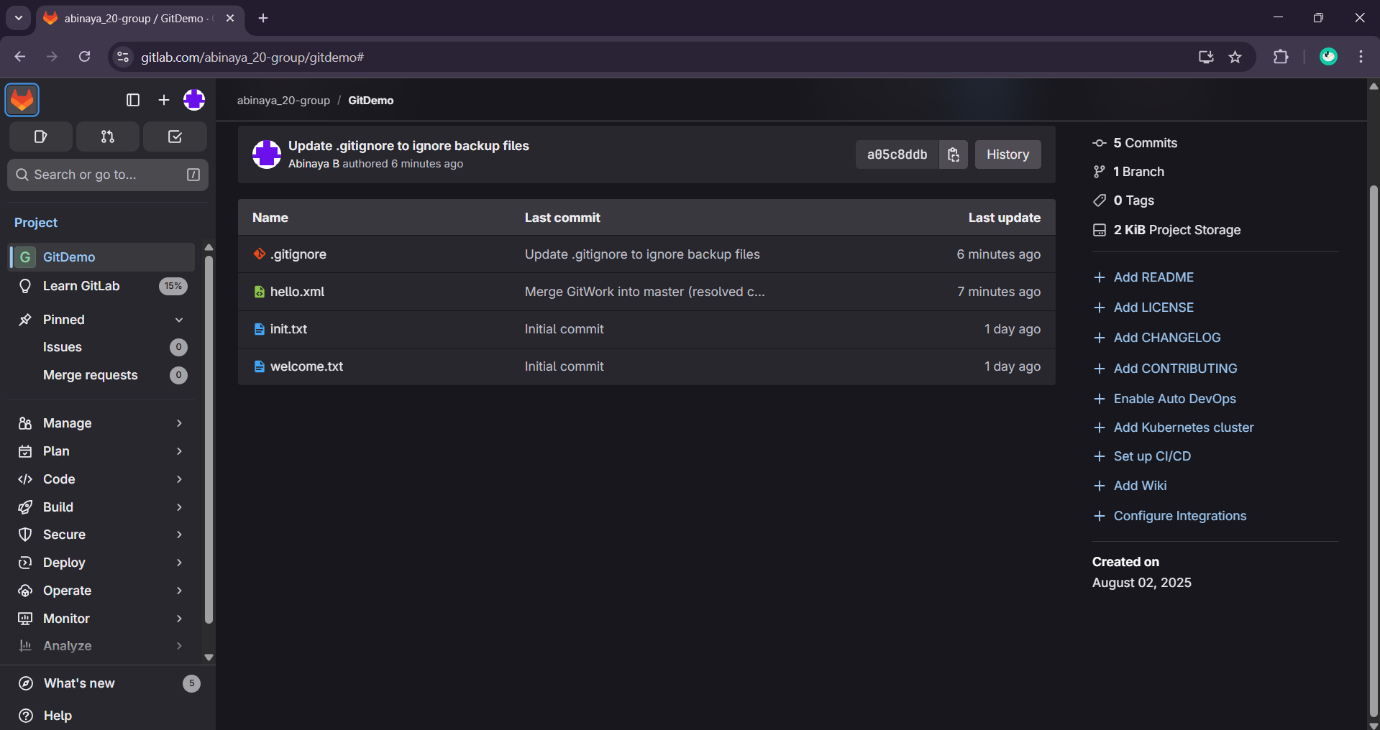
1. **Delete the merged branch**

git branch -d GitWork  
A screenshot of a computer

AI-generated content may be incorrect.

1. **Observe the log**

git log --oneline --graph --decorate



**Exercise 5:**

**Step 1: Verify if master is in clean state**

git checkout master

git status

**Step 2: List out all the available branches**

git branch -a

**Step 3: Pull the remote git repository to the master**

git pull origin master

**Step 4: Push the changes from “Git-T03-HOL\_002” to the remote repository**

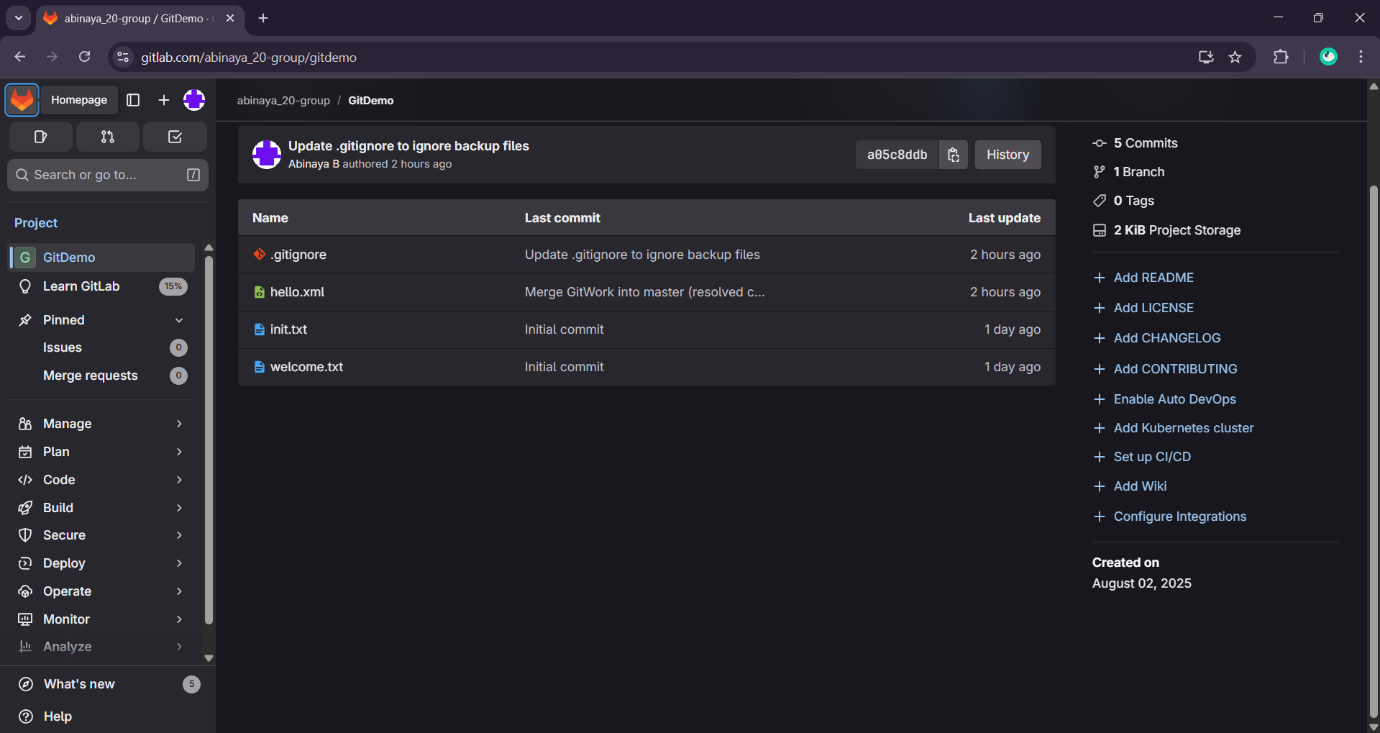
git checkout Git-T03-HOL\_002

git push origin Git-T03-HOL\_002  
**A computer screen shot of a computer screen

AI-generated content may be incorrect.**

**Step 5: Observe if the changes are reflected in the remote repository**

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

****