**GIT**

**EX\_1) Hands-on**

**Step 1 — Setup Git Configuration**

**Check if Git is installed**

git --version

f you see the version (e.g., git version 2.42.0), Git is installed.

**Configure your name and email (user-level)**

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

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

**Verify your configuration**

git config --global --list

You should see:

user.name=Your Name

user.email=your.email@example.com

## ****Step 2 — Set Notepad++ as Default Git Editor****

**Check if Notepad++ runs from Git Bash**

notepad++

If you get an error like command not found, add it to your PATH.

**Add Notepad++ to PATH (Windows)**

Go to **Control Panel → System → Advanced system settings**

Click **Environment Variables**

In **User variables**, find Path → **Edit**

Add the folder path to notepad++.exe (e.g., C:\Program Files\Notepad++)

Click **OK**

**Reopen Git Bash** and check again:

notepad++

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

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

**Verify the editor setting**

git config --global -e

This should open your Git global config in Notepad++.

**Step 3 — Create Local Repository and Add File**

**Create a folder for your project and initialize Git**

mkdir GitDemo

cd GitDemo

git init

**Verify repository creation**

ls -a

You should see a .git folder.

**Create a file and add content**

echo "Welcome to Git Demo!" > welcome.txt

**Check the file**

cat welcome.txt

**Check Git status**

git status

You’ll see welcome.txt as **untracked**.

**Track the file**

git add welcome.txt

**Commit the file (opens Notepad++ for multi-line message)**

git commit

In Notepad++, type your commit message, save, and close.

**Step 4 — Link to Remote Repository and Push**

**On GitLab**, create a new project named **GitDemo** (without a README).

**Add remote repository in Git Bash**

git remote add origin

https://gitlab.com/<your-username>/GitDemo.git

**Pull remote master branch (if needed)**

git pull origin master --allow-unrelated-histories

**Push your local commits to GitLab**

git push -u origin master

EX-2) Hands-on

**1 — Create** .log **file and** log/ **folder**

# Make sure you're inside your Git repository

cd path/to/GitDemo

# Create a .log file

echo "This is a sample log file." > debug.log

# Create log folder and a file inside it

mkdir log

echo "Application started" > log/app.log

## ****2 — Create / Update**** .gitignore

# Create a .gitignore file if it doesn't exist

touch .gitignore

# Add rules to ignore .log files and log/ folder

echo "\*.log" >> .gitignore

echo "log/" >> .gitignore

\*.log → ignores **all** .log files anywhere in the repository

log/ → ignores the entire log directory and its contents

## ****3 — Verify**** .gitignore ****rules****

cat .gitignore

Expected output:

\*.log

log/

## ****4 — Check Git status****

git status

You should see something like:

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

nothing added to commit but untracked files present (use "git add" to track)

Notice:

.gitignore will show as **untracked** because you haven’t committed it yet.

The .log files and log/ folder **won’t appear** — they’re being ignored.

## ****5 — Commit**** .gitignore

git add .gitignore

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

## ****6 — Re-check status****

git status

Expected:

On branch master

nothing to commit, working tree clean

Working directory → clean (no tracked changes)

Local repository → in sync with working directory

Remote repository → can be updated with git push if you want

Ex-3) Hands-on

## ****Branching****

**1. Create a new branch** GitNewBranch

git branch GitNewBranch

**2. List all local and remote branches**

git branch -a

The \* marks the **current** branch.

Remote branches are prefixed with remotes/.

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

git checkout GitNewBranch

**4. Add some files with content**

echo "File created in GitNewBranch" > branchfile1.txt

echo "Another file for testing" > branchfile2.txt

**5. Commit the changes**

git add branchfile1.txt branchfile2.txt

git commit -m "Add branch-specific files"

**6. Check the branch status**

git status

## ****Merging****

**1. Switch back to** master

git checkout master

**2. List differences between master and GitNewBranch (CLI)**

git diff master GitNewBranch

**3. Visual differences with P4Merge**

git difftool master GitNewBranch

**4. Merge** GitNewBranch **into** master

git merge GitNewBranch

**5. View merge log graphically**

git log --oneline --graph --decorate

**6. Delete the merged branch**

git branch -d GitNewBranch

**7. Check status**

git status

Ex-4) Hands-on

## ****Step-by-step Commands****

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

git checkout master

git status

**2. Create branch** GitWork **and add** hello.xml

git branch GitWork

git checkout GitWork

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

**3. Update** hello.xml **and observe status**

echo "<update>Branch-specific change</update>" >> hello.xml

git status

**4. Commit changes in branch**

git add hello.xml

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

**5. Switch to master**

git checkout master

**6. Add a** hello.xml **to master with different content**

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

**7. Commit changes to master**

git add hello.xml

git commit -m "Add hello.xml to master with different content"

**8. Observe log (all branches)**

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

**9. Check differences between master and branch**

git diff master GitWork

**10. Use P4Merge for visual diff**

git difftool master GitWork

**11. Merge branch into master**

git merge GitWork

**12. Observe Git merge markup**

cat hello.xml

**13. Use 3-way merge tool (P4Merge) to resolve conflict**

git mergetool

Save the resolved file from the merge tool.

**14. Commit resolved conflict**

git add hello.xml

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

**15. Observe status and add backup file to** .gitignore

git status

echo "\*.bak" >> .gitignore

**16. Commit** .gitignore **changes**

git add .gitignore

git commit -m "Add .bak files to .gitignore"

**17. List all available branches**

git branch -a

**18. Delete merged branch**

git branch -d GitWork

**19. Observe final log**

git log --oneline --graph --decorate

Ex-5) Hands-on

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

git checkout master

git status

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

git branch -a

**3. Pull the remote git repository to the master**

git pull origin master

**4. Push pending changes from** Git-T03-HOL\_002 **to the remote repository**

git checkout Git-T03-HOL\_002

git push origin Git-T03-HOL\_002

**5. Observe if the changes are reflected in the remote repository**

This step is done by visiting the repository page in your Git hosting service (e.g., GitLab or GitHub) and checking the branch contents.