Week-8

# GIT

## Exercise-1

**Familiar with Git commands like git init, git status, git add, git commit, git push, and git pull.**

1. Verify Git Installation

git –version

1. Set Global User Configuration

git config --global user.name "Gali-Iswarya"

git config --global user.email "239x5a05l4@gprec.ac.in"

1. Verify Configuration

git config --global –list

**1. Check if Notepad++ is Recognized**

notepad++

If it says “command not found”, add it to **System Environment Variables**.

**Steps to add path:**

* Go to **Control Panel > System > Advanced system settings**
* Click **Environment Variables**
* Under **User Variables > Path > Edit**, add path:

C:\Program Files\Notepad++\

**2. Restart Git Bash and Check Again**

notepad++

**3. Create Alias for Notepad++**

alias np='"/c/Program Files/Notepad++/notepad++.exe"'

**4. Set Notepad++ as Git Default Editor**

git config --global core.editor "'/c/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"

**5. Confirm Editor Setting**

git config --global –edit

**Step 3: Create and Add File to Git Repository**

**1. Create Git Repository**

mkdir GitDemo

cd GitDemo

git init

**2. Check Hidden Git Files**

ls -a

**3. Create File and Add Content**

echo "Welcome to Git training session!" > welcome.txt

**4. Verify File Created**

ls

**5. Check File Content**

cat welcome.txt

**6. Check Git Status**

git status

**7. Add File to Staging Area**

git add welcome.txt

**8. Commit with Multi-Line Message**

git commit

Notepad++ will open. Enter a message like:

Added welcome.txt file

- Contains welcome message

- Part of GitDemo repository

Save and close.

**9. Check Status Again**

git status

It should say “nothing to commit, working tree clean”

**Step 4: Push to GitLab**

**1. Create Remote Repo on GitLab**

* Name: **GitDemo**
* Copy the repo URL (e.g., https://gitlab.com/Gali-Iswarya/GitDemo.git)

**2. Add Remote to Local Repo**

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

**4. Push Local Repo to Remote**

git push -u origin master

## Exercise-2

**Explain how to ignore unwanted files using git ignore**

**Step 1: Open Your Git Project Directory**

cd GitDemo

**Step 2: Create Unwanted Files and Folders**

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

mkdir log

echo "Another log file" > log/app.log

Verify the structure:

ls

ls log

**Step 3: Check Git Status (Before Ignoring)**

git status

**Step 4: Create a .gitignore File**

Create and open .gitignore:

notepad++ .gitignore

Add the following lines:

\*.log

log/

**Step 5: Check Git Status Again**

git status

**Step 6: Stage and Commit .gitignore**

git add .gitignore

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

**Step 7: Push to Remote Repository**

git push origin master

## Exercise-3

**Construct a branch, do some changes in the branch, and merge it with master**

**Part 1: Branching**

**Create a new branch**

git checkout -b GitNewBranch

This command both creates and switches to the new branch.

**List all branches**

git branch -a

* Local branches are shown in white.
* Remote branches are shown in red.
* The current branch has a \* next to it.

**Add files and content to new branch**

echo "This is a feature file." > feature.txt

git add feature.txt

**Commit changes to the branch**

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

**Check status**

git status

You should see a clean working directory.

**Part 2: Merging**

**Switch back to the master branch**

git checkout master

**List the differences between master and GitNewBranch**

git diff master..GitNewBranch

**Configure P4Merge**

git config --global merge.tool p4merge

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

**Run P4Merge to visually see differences:**

git difftool master..GitNewBranch

**Merge the branch into master**

git merge GitNewBranch

git commit -m "Merged GitNewBranch into master"

**View the merge history (graph view)**

git log --oneline --graph –decorate

**Delete the merged branch**

git branch -d GitNewBranch

git status

## Exercise-4

**Implement conflict resolution when multiple users are updating the trunk (or master) in such a way that it results into a conflict with the branch’s modification.**

**Step 1: Verify Clean Working Directory**

cd Git-T03-HOL\_001

git status

You should see:

nothing to commit, working tree clean

**Step 2: Create Branch GitWork and Add hello.xml**

git checkout -b GitWork

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

**Step 3: Modify hello.xml**

echo "<note>This is branch content</note>" >> hello.xml

git status

**Step 4: Commit the File**

git add hello.xml

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

**Step 5: Switch Back to master**

git checkout master

**Step 6: Add a Conflicting hello.xml**

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

echo "<note>This is master content</note>" >> hello.xml

**Step 7: Commit Changes to Master**

git add hello.xml

git commit -m "Added hello.xml in master with different content"

**Step 8: View Commit History**

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

**Step 9: Compare Master and Branch**

git diff master..GitWork

**Step 10: Visual Diff with P4Merge**

git config --global merge.tool p4merge

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

git difftool master..GitWork

**Step 11: Try Merging GitWork into master**

git merge GitWork

**Step 12: Observe Git Conflict Markup**

Open the file:

notepad++ hello.xml

**<<<<<<< HEAD**

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

**<note>This is master content</note>**

**=======**

**<message>Hello from GitWork branch</message>**

**<note>This is branch content</note>**

**>>>>>>> GitWork**

**Step 14: Finalize the Merge**

git add hello.xml

git commit -m "Resolved conflict between master and GitWork in hello.xml"

**Step 15: Add Merge Backup File to .gitignore**

echo "\*.orig" >> .gitignore

**Step 16: Commit .gitignore Changes**

git add .gitignore

git commit -m "Ignore merge conflict backup files"

**Step 17: List All Branches**

git branch

**Step 18: Delete Merged Branch**

git branch -d GitWork

**Step 19: View Final Merge Graph**

git log --oneline --graph --decorate

## Exercise-5

**Execute steps involving clean up and push back to remote Git.**

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

git status

* Output should show:  
  nothing to commit, working tree clean  
  This means your working directory is clean with no pending changes.

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

git branch -a

* This shows all **local** and **remote** branches.  
  \* master

feature/login

remotes/origin/master

remotes/origin/feature/login

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

git checkout master

git pull origin master

* This ensures your local master is **up to date** with the latest changes from the remote.

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

git push origin master

* This pushes your local master branch changes to the remote master branch.

**Step 5: Verify that changes are reflected in remote repository**

* Go to your **GitHub/GitLab** repository in a browser.
* Check:
  + Latest commit timestamp.
  + Recent file changes.
  + Push log.