**1. Git-HOL**

Step 1 – Git Configuration

Check if Git is installed

git --version

Expected Output:

Set your username and email

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

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

Verify the configuration

git config --list

Expected Output:

user.name=Your Name

user.email=your\_email@example.com

...

Step 2 – Set Notepad++ as Default Git Editor

Check if Notepad++ opens from Git Bash

notepad++

If you see:

bash: notepad++: command not found

→ You must add Notepad++ to the Windows PATH.

Add Notepad++ path to environment variables

Go to Control Panel → System → Advanced System Settings

Click Environment Variables

Edit the Path variable → Add:

C:\Program Files\Notepad++\

Restart Git Bash.

Verify

notepad++

Expected Result: Notepad++ opens.

Create a short alias for Notepad++

alias np='notepad++'

Set Notepad++ as the default Git editor

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

Check editor configuration

git config --global -e

Expected Result: Notepad++ opens showing the Git global config.

Step 3 – Create Repository and Add Files

Create a new folder & initialize Git

mkdir GitDemo

cd GitDemo

git init

Expected Output:

Initialized empty Git repository in <path>/GitDemo/.git/

Create a file

echo "Welcome to Git Hands-on Lab" > welcome.txt

Check if file exists

ls

Expected Output:

welcome.txt

View file content

cat welcome.txt

Expected Output:

Welcome to Git Hands-on Lab

Check repository status

git status

Expected Output:

Untracked files:

welcome.txt

Stage the file

git add welcome.txt

Commit changes (opens Notepad++ for message)

git commit

Expected Result:

Notepad++ opens → Type commit message → Save & close.

Git shows:

[master (root-commit) abc1234] Your commit message

1 file changed, 1 insertion(+)

create mode 100644 welcome.txt

Step 4 – Push to Remote Repository

Add remote

git remote add origin <your\_gitlab\_repo\_url>

Pull from remote (optional)

git pull origin master

Push local repo to remote

git push origin master

**2. Git-HOL**

Step 1 – Create Files and Folders to Ignore

# Create a folder and file

mkdir log

touch log/debug.log

touch error.log

# Check if files exist

ls

Expected Output:

error.log log/

ls log

Expected Output:

debug.log

Step 2 – Create .gitignore File

# Create .gitignore

touch .gitignore

# Open and edit .gitignore

notepad++ .gitignore

Add the following lines to .gitignore:

\*.log

log/

Save and close the file.

Step 3 – Check Git Status Before Commit

git status

Expected Output:

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

(Notice that error.log and log/ are NOT shown — they are ignored.)

Step 4 – Add and Commit .gitignore

git add .gitignore

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

Expected Output:

[master abc1234] Added .gitignore to ignore .log files and log folder

1 file changed, 2 insertions(+)

create mode 100644 .gitignore

Step 5 – Verify Ignored Files

git status

Expected Output:

On branch master

nothing to commit, working tree clean

**3. Git-HOL**

Step 1 – Create and Work on a New Branch

# Create a new branch

git branch GitNewBranch

# List all branches (local & remote if any)

git branch -a

Expected Output:

markdown

\* master

GitNewBranch

(The \* shows the current branch — here it's master.)

# Switch to the new branch

git checkout GitNewBranch

Expected Output:

Switched to branch 'GitNewBranch'

# Add a new file in the branch

echo "This is a new branch file" > branch\_file.txt

# Stage and commit changes

git add branch\_file.txt

git commit -m "Added branch\_file.txt in GitNewBranch"

Expected Output:

[GitNewBranch abc1234] Added branch\_file.txt in GitNewBranch

1 file changed, 1 insertion(+)

create mode 100644 branch\_file.txt

# Check status

git status

Expected Output:

On branch GitNewBranch

nothing to commit, working tree clean

Step 2 – Merge the Branch into Master

# Switch back to master

git checkout master

Expected Output:

Switched to branch 'master'

Your branch is up to date with 'origin/master'.

# List differences between master and branch

git diff master GitNewBranch

Expected Output (example):

diff --git a/branch\_file.txt b/branch\_file.txt

new file mode 100644

index 0000000..fcf5d45

--- /dev/null

+++ b/branch\_file.txt

@@

+This is a new branch file

# (Optional) Use P4Merge for visual diff

git difftool master GitNewBranch

Expected Result:

P4Merge opens showing file differences visually.

# Merge branch into master

git merge GitNewBranch

Expected Output:

Updating 12ab34c..56cd78e

Fast-forward

branch\_file.txt | 1 +

1 file changed, 1 insertion(+)

create mode 100644 branch\_file.txt

Step 3 – Check Merge History

git log --oneline --graph --decorate

Expected Output (example):

\* 56cd78e (HEAD -> master, GitNewBranch) Added branch\_file.txt in GitNewBranch

\* 12ab34c Initial commit

Step 4 – Delete the Branch

git branch -d GitNewBranch

Expected Output:

Deleted branch GitNewBranch (was 56cd78e).

# Check status after deleting branch

git status

Expected Output:

On branch master

nothing to commit, working tree clean

**4. Git-HOL**

Step 1 – Verify Master is Clean

git checkout master

git status

Expected Output:

On branch master

nothing to commit, working tree clean

Step 2 – Create Branch and Commit Changes

# Create and switch to new branch

git checkout -b GitWork

Expected Output:

Switched to a new branch 'GitWork'

# Create hello.xml in branch

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

# Stage and commit

git add hello.xml

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

Expected Output:

[GitWork abc1234] Added hello.xml in GitWork

1 file changed, 1 insertion(+)

create mode 100644 hello.xml

Step 3 – Create Different File in Master

git checkout master

Expected Output:

Switched to branch 'master'

# Create conflicting hello.xml in master

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

git add hello.xml

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

Expected Output:

[master def5678] Added hello.xml in master

1 file changed, 1 insertion(+)

create mode 100644 hello.xml

Step 4 – View Log

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

Expected Output (example):

\* abc1234 (GitWork) Added hello.xml in GitWork

| \* def5678 (HEAD -> master) Added hello.xml in master

|/

\* 89abcde Initial commit

Step 5 – Compare Branches

git diff master GitWork

Expected Output:

diff --git a/hello.xml b/hello.xml

--- a/hello.xml

+++ b/hello.xml

@@

-<message>Hello from master</message>

+<message>Hello from branch</message>

(Optional visual diff):

git difftool master GitWork

Expected Result: P4Merge shows the difference visually.

Step 6 – Merge and Resolve Conflict

git merge GitWork

Expected Output:

Auto-merging hello.xml

CONFLICT (content): Merge conflict in hello.xml

Automatic merge failed; fix conflicts and then commit the result.

# Open in merge tool

git mergetool

Expected Result:

P4Merge (or your 3-way merge tool) opens showing both versions.

Resolve conflict, choose the correct lines, save, and close.

Step 7 – Commit the Resolution

git add hello.xml

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

Expected Output:

[master ghi9012] Resolved merge conflict in hello.xml

Step 8 – Ignore Backup Files

echo "\*.orig" >> .gitignore

git add .gitignore

git commit -m "Ignore merge backup files"

Expected Output:

[master jkl3456] Ignore merge backup files

1 file changed, 1 insertion(+)

Step 9 – Delete Branch

git branch -d GitWork

Expected Output:

Deleted branch GitWork (was abc1234).

Step 10 – Final Log

git log --oneline --graph --decorate

Expected Output (example):

\* jkl3456 (HEAD -> master) Ignore merge backup files

\* ghi9012 Resolved merge conflict in hello.xml

\* def5678 Added hello.xml in master

\* abc1234 Added hello.xml in GitWork

\* 89abcde Initial commit

**5. Git-HOL**

Step 1 – Verify Master is Clean

git checkout master

git status

Expected Output:

On branch master

nothing to commit, working tree clean

Step 2 – List All Available Branches

git branch -a

Expected Output (example):

\* master

GitWork

remotes/origin/master

Step 3 – Pull Remote Repository to Master

git pull origin master

Expected Output (example):

Already up to date.

(Or if there are updates, Git will fetch and merge them.)

Step 4 – Push Pending Changes to Remote

git push origin master

Expected Output:

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 4 threads

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 321 bytes | 321.00 KiB/s, done.

Total 3 (delta 1), reused 0 (delta 0), pack-reused 0

To https://gitlab.com/yourusername/yourrepo.git

abc1234..def5678 master -> master

Step 5 – Verify Changes in Remote Repository

You can check:

git log --oneline --decorate -n 5

Expected Output (example):

def5678 (HEAD -> master, origin/master) Commit message from Git-T03-HOL\_002

abc1234 Previous commit message

...