**Git Hands-On Lab – Step-by-Step Guide**

**Objective**

By the end of this lab, you will be able to:

* Initialize a Git repository
* Track files and commit changes
* Configure Git settings
* Push and pull from a remote repository
* Use **Notepad++** as the default Git editor

**Prerequisites**

1. **Install Git Bash** from <https://git-scm.com/downloads>
2. **Install Notepad++** from <https://notepad-plus-plus.org/>
3. **Create a GitHub or GitLab account** (don’t use company credentials)

**Step 1 – Setup Git Configuration**

**1. Check Git installation**

git --version

**Expected Output:**

git version 2.x.x

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

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

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

**3. Verify Git configuration**

git config --list

**Expected Output:**

user.name=Your Name

user.email=your.email@example.com

**Step 2 – Integrate Notepad++ as the Default Git Editor**

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

notepad++

* If you get **command not found**, add it to your PATH.

**2. Add Notepad++ to environment variables (Windows)**

1. Go to **Control Panel → System → Advanced system settings**
2. Click **Environment Variables**
3. Edit the **Path** variable under "User variables"
4. Add the folder path where notepad++.exe is located (e.g., C:\Program Files\Notepad++)

**3. Restart Git Bash and check again**

notepad++

Notepad++ should now open.

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

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

**5. Verify**

git config --global -e

This will open your Git config file in Notepad++.

**Step 3 – Create a Local Repository and Add a File**

**1. Create a new project folder**

mkdir GitDemo

cd GitDemo

**2. Initialize Git**

git init

**Expected Output:**

Initialized empty Git repository in C:/path/to/GitDemo/.git/

**3. Create a file**

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

**4. Verify file creation**

ls

**Expected Output:**

welcome.txt

**5. Check file contents**

cat welcome.txt

**Expected Output:**

Welcome to Git Hands-On Lab

**6. Check status**

git status

**Expected Output:**

Untracked files:

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

welcome.txt

**7. Stage the file**

git add welcome.txt

**8. Commit the file**

git commit -m "Initial commit with welcome.txt"

or, to use Notepad++ for a multi-line commit message:

git commit

(Then write the message in Notepad++ and save.)

**9. Verify commit**

git log

**Step 4 – Push to Remote Repository**

**1. Create a remote repository**

On GitHub or GitLab, create a new repo named GitDemo (without README).

**2. Link local to remote**

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

**3. Push your commit**

git push -u origin master

**4. Pull updates (if needed)**

git pull origin master

**You now have a local Git repository linked to a remote, with Notepad++ as your default editor.**

Got it — your second document is another Git Hands-On Lab, but this time it’s about **.gitignore**.

**Git Hands-On Lab – .gitignore**

**Objective**

By the end of this lab, you will:

* Understand what .gitignore does
* Ignore specific files and folders from being tracked in Git

**Prerequisites**

* Git installed and configured (git --version)
* Notepad++ integrated as the default Git editor
* Local Git repository already initialized and connected to a remote (from previous HOL)

**Step 1 – Create Files and Folders to Ignore**

1. **Navigate to your project folder**

cd GitDemo

1. **Create a .log file**

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

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

mkdir log

echo "Some log content" > log/system.log

1. **Check current status**

git status

**Expected Output (part):**

Untracked files:

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

debug.log

log/

**Step 2 – Create a .gitignore File**

1. **Open .gitignore in Notepad++**

notepad++ .gitignore

1. **Add the following lines to ignore .log files and the log folder:**

\*.log

log/

* \*.log → ignores all files ending with .log
* log/ → ignores the log folder and its contents

1. **Save and close the file.**

**Step 3 – Verify .gitignore Behavior**

1. **Check status again**

git status

**Expected Output:**

Untracked files:

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

.gitignore

Notice that debug.log and log/ are no longer listed.

1. **Add .gitignore to staging**

git add .gitignore

1. **Commit the .gitignore file**

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

**Step 4 – Push Changes to Remote**

git push origin master

**Git Hands-On Lab – Branching & Merging**

**Objective**

By the end of this lab, you will:

* Create and work in a new branch
* Make changes in that branch
* Merge branch changes into master (or main)
* View differences in command-line and P4Merge

**Prerequisites**

* Git installed and configured
* Notepad++ set as default editor
* P4Merge installed (optional for visual diff) → <https://www.perforce.com/downloads/visual-merge-tool>
* Local repository already created and linked to remote

**Step 1 – Create a New Branch**

1. **Ensure you’re in your project**

cd GitDemo

1. **Create a branch named GitNewBranch**

git branch GitNewBranch

1. **List branches**

git branch -a

**Expected Output:**

\* master

GitNewBranch

(\* marks the current branch)

**Step 2 – Switch to the New Branch**

git checkout GitNewBranch

**Expected Output:**

Switched to branch 'GitNewBranch'

**Step 3 – Add Files in the New Branch**

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

git add branchfile.txt

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

**Step 4 – Check Status**

git status

**Step 5 – Switch Back to Master**

git checkout master

**Step 6 – View Differences (Command-Line)**

git diff master GitNewBranch

**Step 7 – View Differences in P4Merge (Optional)**

git mergetool

* If configured, P4Merge will open to show visual differences.

**Step 8 – Merge the Branch into Master**

git merge GitNewBranch

**Step 9 – View Merge History**

git log --oneline --graph --decorate

**Step 10 – Delete the Merged Branch**

git branch -d GitNewBranch

**Expected Output:**

Deleted branch GitNewBranch (was <commit-hash>).

**You have successfully created a branch, made changes, and merged it into master.**

**Git Hands-On Lab – Merge Conflict Resolution**

**Objective**

By the end of this lab, you will:

* Simulate a merge conflict between a branch and master
* Use command-line and P4Merge (optional) to resolve it
* Update .gitignore for backup files after merge

**Prerequisites**

* Git installed and configured
* Notepad++ set as default editor
* P4Merge installed for visual merge (optional)
* Local repo with a master branch in a clean state

**Step 1 – Verify Master is Clean**

git checkout master

git status

✅ Should show:

nothing to commit, working tree clean

**Step 2 – Create a Branch**

git branch GitWork

git checkout GitWork

**Step 3 – Add a File in Branch**

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

git add hello.xml

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

**Step 4 – Modify the File in Branch**

echo "<message>Updated in branch</message>" > hello.xml

git add hello.xml

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

**Step 5 – Switch to Master and Create Same File**

git checkout master

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

git add hello.xml

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

**Step 6 – View History**

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

**Step 7 – Check Differences**

git diff master GitWork

(Optional visual diff with P4Merge):

git mergetool

**Step 8 – Merge and Trigger 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.

**Step 9 – Resolve Conflict**

1. Open hello.xml in Notepad++ or use git mergetool.
2. Remove conflict markers (<<<<<<<, =======, >>>>>>>) and keep the correct version or merge both changes.

Example final file:

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

1. Stage resolved file:

git add hello.xml

1. Commit resolution:

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

**Step 10 – Update .gitignore for Backup Files**

If merge tools created backup files (e.g., hello.xml.orig):

echo "\*.orig" >> .gitignore

git add .gitignore

git commit -m "Ignore merge backup files"

**Step 11 – Delete Merged Branch**

git branch -d GitWork

**Step 12 – View Final History**

git log --oneline --graph --decorate

**Git Hands-On Lab – Cleanup & Push to Remote**

**Objective**

By the end of this lab, you will:

* Ensure your master branch is clean
* Pull the latest changes from remote
* Push pending changes to remote
* Verify updates on the remote repository

**Prerequisites**

* Git installed and configured
* Remote repository linked
* Prior lab changes from **Git-T03-HOL\_002** available locally

**Step 1 – Verify Master is Clean**

git checkout master

git status

✅ Should show:

nothing to commit, working tree clean

If not clean:

* Stage and commit changes (git add . && git commit -m "message")
* Or stash them (git stash)

**Step 2 – List All Branches**

git branch -a

Example Output:

\* master

feature-branch

remotes/origin/master

**Step 3 – Pull Latest Changes from Remote**

git pull origin master

This ensures your local master is up to date with the remote repository.

**Step 4 – Push Pending Changes to Remote**

If you have commits from **Git-T03-HOL\_002**:

git push origin master

**Step 5 – Verify on Remote**

* Go to your remote repository (GitHub/GitLab) in a browser.
* Confirm your latest commit(s) appear in the **master** branch.

**You have successfully cleaned up your master branch, synced with remote, and pushed changes.**