## **GIT\_HandsOn\_Week8**

## **Objectives**

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

In this hands-on lab, you will learn how to

* Setup your machine with Git Configuration
* Integrate notepad++.exe to Git and make it a default editor
* Add a file to source code repository

## **Prerequisites**

* Install Git Bash client in your machine

Notes\*:

|  |
| --- |
| Please follow the below steps for creating a free account in GitHub.  Don’t use cognizant credentials to login to GitHub. |

Estimated time to complete this lab: **30 minutes.**

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

To create a new repository, signup with GitLab and register your credentials

Login to GitLab and create a “GitDemo” project

1. To check if Git client is installed properly: Open Git bash shell and execute



If output shows Git with its version information that indicates, that Git Client installs properly.

1. To configure user level configuration of user ID and email ID execute



1. To check if the configuration is properly set, execute the following command.



**Step 2: Integrate notepad++.exe to Git and make it a default editor**

1. To check, if notepad++.exe execute from Git bash



If Git bash could not able to recognize notepad++ command that implies notepad++.exe is note added to the environment path variable.

To add path of notepad++.exe to environment variable, go to control panel -> System -> Advanced System settings. Go to Advanced tab -> Environment variables -> Add path of notepad++.exe to the path user variable by clicking on “Edit”



1. Exit Git bash shell, open bash shell and execute



Now, notepad++ will open from Git bash shell

1. To create an alias command for notepad++.exe, execute



It will open notepad++ from bash shell, and create a user profile by adding the line in notepad++



1. To configure the editor, execute the command



1. To verify if notepad++ is the default editor, execute the command



Here ‘-e’ option implies editor

It will show the entire global configuration as shown below,



**Step 3: Add a file to source code repository**

1. Open Git bash shell and create a new project “**GitDemo**” by executing the command



1. Git bash initializes the “**GitDemo**” repository. To verify, execute the command



It will display all the hidden files in the Git “working directory”.

1. To create a file **“welcome.txt”** and add content to the file, execute the command



1. To verify if the file “welcome.txt” is created, execute



1. To verify the content, execute the command



1. Check the status by executing



Now the file **“welcome.txt”** is available in Git “working directory”

1. To make the file to be tracked by Git repository, execute the command



1. To add multi line comments, we are opening default editor to comment. Execute the command



Notepad++ editor will open and to add multi-line comment with default editor

1. To check if local and “Working Directory” git repository are same, execute git status



**welcome.txt** is added to the local repository.

1. Signup with GitLab and create a remote repository **“GitDemo”**
2. To pull the remote repository, execute

git pull origin master

1. To push the local to remote repository, execute

git push origin master

**Steps**:

1. **Verify Git Installation**:
   * Open Git Bash and run:

git --version

* + Expected output: Git version (e.g., git version 2.45.2). If no version appears, reinstall Git.

1. **Configure Git User**:
   * Set your user name and email (use your personal GitLab credentials):

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

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

* + Verify configuration:

git config --global --list

Output should show user.name and user.email.

1. **Integrate Notepad++ as Default Editor**:
   * Verify Notepad++ is accessible:

notepad++

If it fails, add Notepad++ to PATH:

* + - Go to Control Panel → System → Advanced System Settings → Environment Variables.
    - Edit the Path variable, add the Notepad++ directory (e.g., C:\Program Files\Notepad++).
    - Reopen Git Bash and test again.
  + Set Notepad++ as the default editor:

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

* + Verify:

git config --global --list

Look for core.editor=notepad++ -multiInst -nosession.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Create and Initialize a Repository**:
   * Create a directory for the project and initialize a Git repository:

mkdir GitDemo

cd GitDemo

git init

* + Verify initialization (check for .git folder):

ls -a

Output should include .git.

A computer screen shot of a black square with white text

AI-generated content may be incorrect.

1. **Add a File to the Repository**:
   * Create a file welcome.txt:

echo "Hello, Git!" > welcome.txt

* + Verify file creation:

ls

cat welcome.txt

Output: welcome.txt and its content (Hello, Git!).

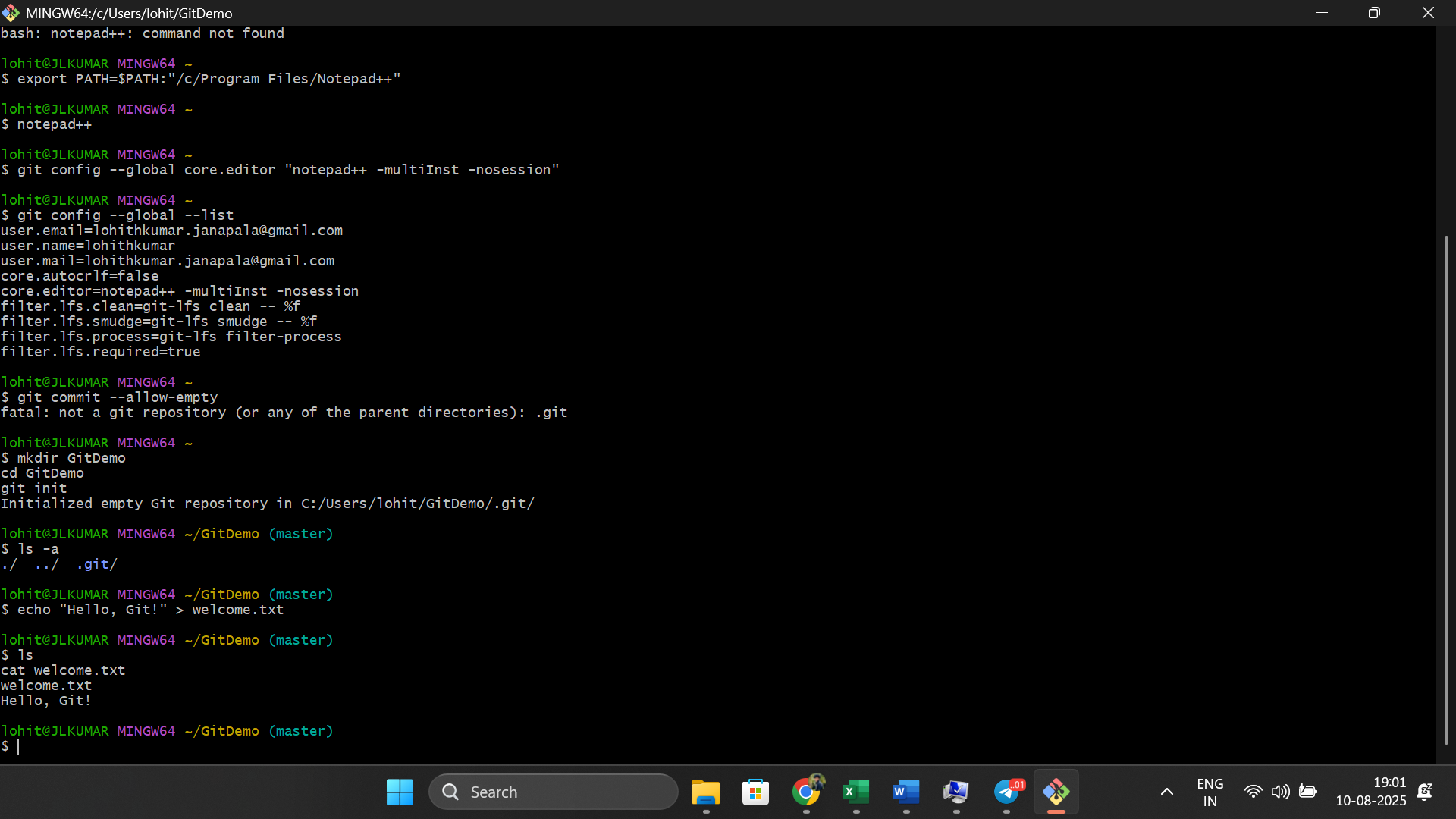
* + Check Git status:

git status

Output: welcome.txt is untracked.

* + Stage the file:

git add welcome.txt



* + Commit with Notepad++:

git commit

Notepad++ opens; add a commit message (e.g., Initial commit), save, and close.

* + Verify status:

git status

Output: Working directory clean.

1. **Set Up Remote Repository**:
   * Create a GitDemo repository on GitLab (via GitLab UI, not using Cognizant credentials).
   * Link local repository to remote:

git remote add origin <GitLab repository URL>

Replace <GitLab repository URL> with the HTTPS/SSH URL from GitLab.

* + Pull from remote (if initialized with a README):

git pull origin master

Resolve any merge conflicts if prompted (unlikely for a new repo).

* + Push to remote:

git push origin master

**Key Concepts**:

* git init: Initializes a new Git repository.
* git add: Stages files for commit.
* git commit: Saves staged changes with a message.
* git push/git pull: Syncs local and remote repositories.
* Notepad++ integration simplifies editing commit messages.

### **2. Git-HOL.docx: Using .gitignore**

**Objective**: Learn to use .gitignore to ignore unwanted files (.log files and log folders).

**Prerequisites**:

* Git environment set up (from Lab 1).
* Notepad++ integrated.
* Local and remote Git repository (GitDemo).

**Steps**:

1. **Create Files to Ignore**:
   * In the GitDemo directory:

echo "Log content" > test.log

mkdir log

echo "Another log" > log/another.log

1. **Create/Update .gitignore**:
   * Create .gitignore:

echo "\*.log" > .gitignore

echo "log/" >> .gitignore

* + - \*.log: Ignores all files with .log extension.
    - log/: Ignores the log folder and its contents.
  + Verify .gitignore content:

cat .gitignore

A screenshot of a computer

AI-generated content may be incorrect.

1. **Check Git Status**:
   * Run:

git status

Output: Only .gitignore should appear as untracked; test.log and log/ are ignored.

1. **Commit .gitignore**:
   * Stage and commit:

git add .gitignore

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

1. **Verify Status**:
   * Check again:

git status

Output: Working directory clean (no .log or log/ files).

1. **Push to Remote** (optional, not specified but good practice):

git push origin master

**Key Concepts**:

* .gitignore: Specifies files/folders Git should ignore (e.g., logs, temporary files).
* Patterns like \*.log match all files with the specified extension; folder/ matches entire directories.

### **3. Git-HOL.docx: Branching and Merging**

**Objective**: Create a branch, make changes, merge it with master, and create a merge request in GitLab.

**Prerequisites**:

* Git environment with P4Merge tool installed.
* GitLab repository (GitDemo).

**Steps**:

1. **Create a Branch**:
   * Create and switch to GitNewBranch:

git checkout -b GitNewBranch

1. **List Branches**:
   * Run:

git branch -a

Output: Lists all branches; \* GitNewBranch indicates the current branch.

1. **Add Files to Branch**:
   * Create a file:

echo "Branch content" > branch\_file.txt

* + Stage and commit:

git add branch\_file.txt

git commit -m "Add branch\_file.txt to GitNewBranch"

A screenshot of a computer

AI-generated content may be incorrect.

1. **Check Status**:
   * Run:

git status

Output: Working directory clean.

1. **Switch to Master**:

git checkout master

1. **Compare Branches (CLI)**:
   * Check differences:

git diff master GitNewBranch

Output: Shows differences in branch\_file.txt.

1. **Compare with P4Merge**:
   * Configure P4Merge as difftool:

git config --global diff.tool p4merge

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

* + Run:

git difftool master GitNewBranch

P4Merge opens, visually showing differences.

1. **Merge Branch to Master**:
   * Merge:

git merge GitNewBranch

If no conflicts, merge completes.

* + Check log:

git log --oneline --graph --decorate

Output: Shows merge commit.

1. **Delete Branch**:
   * Delete GitNewBranch:

git branch -d GitNewBranch

* + Verify status:

git status

1. **Create Merge Request in GitLab**:
   * Push to remote:

git push origin master

* + In GitLab UI:
    - Go to GitDemo repository.
    - If you pushed GitNewBranch earlier (git push origin GitNewBranch), create a merge request:
      * Navigate to "Merge Requests" → "New Merge Request".
      * Select source (GitNewBranch) and target (master).
      * Submit and merge via GitLab UI.

**Key Concepts**:

* Branching (git checkout -b): Isolates changes.
* Merging (git merge): Combines branch changes into master.
* P4Merge: Visual tool for comparing differences.
* Merge Request: GitLab’s process for reviewing and merging branches.

### **4. Git-HOL.docx: Conflict Resolution**

**Objective**: Resolve merge conflicts when multiple users update master and a branch.

**Prerequisites**:

* Completed Git-T03-HOL\_001 (assumed to be Lab 3).
* P4Merge installed.

**Steps**:

1. **Verify Master is Clean**:
   * git checkout master

git status

Output: Working directory clean.

1. **Create Branch and Add File**:
   * Create GitWork branch:

git checkout -b GitWork

* + Add hello.xml:

echo "<xml>Branch content</xml>" > hello.xml

A screenshot of a computer

AI-generated content may be incorrect.

1. **Update and Commit**:
   * Modify hello.xml:

echo "<xml>Updated branch content</xml>" > hello.xml

* + Check status:

git status

Output: hello.xml modified.

* + Commit:

git add hello.xml

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

1. **Switch to Master and Create Conflict**:

git checkout master

* + Add hello.xml with different content:

echo "<xml>Master content</xml>" > hello.xml

* + Commit:

git add hello.xml

git commit -m "Add hello.xml to master"

1. **View Log**:

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

Output: Shows commits on master and GitWork.

1. **Check Differences**:
   * CLI diff:

git diff master GitWork

* + P4Merge:

git difftool master GitWork

1. **Merge and Resolve Conflict**:
   * Merge GitWork into master:

git checkout master

git merge GitWork

Output: Conflict in hello.xml.

* + Use P4Merge for 3-way merge:

git mergetool

* + - P4Merge opens; resolve conflicts by choosing or combining content from master and GitWork.
    - Save and close P4Merge.
  + Stage resolved file:

git add hello.xml

1. **Commit Merged Changes**:

git commit -m "Resolve merge conflict"

1. **Add Backup File to .gitignore**:
   * If merge created backup files (e.g., hello.xml.orig), add to .gitignore:

echo "\*.orig" >> .gitignore

* + Commit:

git add .gitignore

git commit -m "Ignore backup files"

1. **List and Delete Branch**:
   * List branches:

git branch -a

* + Delete GitWork:

git branch -d GitWork

1. **View Final Log**:

git log --oneline --graph --decorate

**Key Concepts**:

* Merge Conflicts: Occur when same file is modified differently in two branches.
* 3-Way Merge: Compares base, source, and target versions to resolve conflicts.
* git mergetool: Uses tools like P4Merge for visual conflict resolution.

### **5. Git-HOL.docx: Clean Up and Push to Remote**

**Objective**: Clean up local repository and push changes to remote GitLab repository.

**Prerequisites**:

* Completed Git-T03-HOL\_002 (Lab 4).
* GitLab repository (GitDemo).

**Steps**:

1. **Verify Master is Clean**:

A screenshot of a computer

AI-generated content may be incorrect.

* + git checkout master

git status

Output: Working directory clean.

1. **List Branches**:

git branch -a

1. **Pull Remote Repository**:
   * Ensure master is up-to-date:

git pull origin master

1. **Push Changes**:
   * Push changes from Lab 4 (e.g., merged hello.xml, updated .gitignore):

git push origin master

1. **Verify Remote Repository**:
   * In GitLab UI, check the GitDemo repository for updated files (hello.xml, .gitignore).

**Key Concepts**:

* git pull: Syncs local repository with remote.
* git push: Updates remote repository with local commits.
* Always verify status to avoid pushing uncommitted changes.