**Questions :**

**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

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.

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



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**

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”



Exit Git bash shell, open bash shell and execute



Now, notepad++ will open from Git bash shell

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++



To configure the editor, execute the command



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**

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



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



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

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



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



To verify the content, execute the command



Check the status by executing



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

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



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

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



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

Signup with GitLab and create a remote repository **“GitDemo”**

To pull the remote repository, execute

git pull origin master

To push the local to remote repository, execute

git push origin master

2. Create a **“.log”** file and a **log folder** in the working directory of Git. Update the **.gitignore** file in such a way that on committing, these files (.log extensions and log folders) are ignored.

Verify if the git status reflects the same about working directory, local repository and git repository.

3. Please follow the instruction to complete the hands-on. Each instruction expects a command for the Git Bash.

**Branching:**

Create a new branch **“GitNewBranch”.**

List all the local and remote branches available in the current trunk. Observe the “\*” mark which denote the current pointing branch.

Switch to the newly created branch. Add some files to it with some contents.

Commit the changes to the branch.

Check the status with **“git status”** command.

**Merging:**

Switch to the master

List out all the differences between trunk and branch. These provide the differences in command line interface.

List out all the visual differences between master and branch using **P4Merge tool**.

Merge the source branch to the trunk.

Observe the logging after merging using **“git log –oneline –graph –decorate”**

Delete the branch after merging with the trunk and observe the git status.

4. Please follow the instructions to complete the hands-on. Each instruction expect a command for the Git Bash.

Verify if master is in clean state.

Create a branch **“GitWork”.** Add a file “hello.xml”.

Update the content of “hello.xml” and observe the status

Commit the changes to reflect in the branch

Switch to master.

Add a file **“hello.xml”** to the master and add some different content than previous.

Commit the changes to the master

Observe the log by executing **“git log –oneline –graph –decorate –all”**

Check the differences with Git diff tool

For better visualization, use P4Merge tool to list out all the differences between master and branch

Merge the bran to the master

Observe the git mark up.

Use 3-way merge tool to resolve the conflict

Commit the changes to the master, once done with conflict

Observe the git status and add backup file to the .gitignore file.

Commit the changes to the .gitignore

List out all the available branches

Delete the branch, which merge to master.

Observe the log by executing **“git log –oneline –graph –decorate”**

5. Please follow the instructions to complete the hands-on. Each instruction expects a command for the Git Bash.

Verify if master is in clean state.

List out all the available branches.

Pull the remote git repository to the master

Push the changes, which are pending from **“Git-T03-HOL\_002”** to the remote repository.

Observe if the changes are reflected in the remote repository.

Solutions:

# --- 1. Setup Git ---

git --version

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

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

git config --global --list

# After adding Notepad++ path to environment variables in Windows:

alias np='notepad++.exe'

git config --global core.editor "notepad++.exe"

# --- 2. Create repo and add file ---

mkdir GitDemo

cd GitDemo

git init

echo "Welcome to Git Demo" > welcome.txt

git status

git add welcome.txt

git commit # Will open Notepad++ for multi-line commit

# Link to GitLab (replace with your actual GitLab repo URL)

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

git pull origin master

git push origin master

# --- 3. Ignore .log files and log folder ---

mkdir log

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

echo "\*.log" >> .gitignore

echo "log/" >> .gitignore

git status

# --- 4. Branching ---

git branch GitNewBranch

git branch -a

git checkout GitNewBranch

echo "This is new branch content" > branch.txt

git add branch.txt

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

git status

# --- 5. Merging ---

git checkout master

git diff GitNewBranch

git merge GitNewBranch

git log --oneline --graph --decorate

git branch -d GitNewBranch

# --- 6. Conflict Resolution Scenario ---

git checkout -b GitWork

echo "Branch hello" > hello.xml

git add hello.xml

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

git checkout master

echo "Master hello" > hello.xml

git add hello.xml

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

git merge GitWork

# Resolve conflicts manually in hello.xml, then:

git add hello.xml

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

echo "\*.bak" >> .gitignore

git add .gitignore

git commit -m "Updated .gitignore to ignore backup files"

git branch -d GitWork

# --- 7. Remote Repo Sync ---

git branch -a

git pull origin master

git push origin master