**digital nurture Deep skilling-Week8**

## **git-HOL**

## **exercise 1: 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

**COMMANDS AND OUTPUT:**

| **Step** | **Command** | **Expected Output** |
| --- | --- | --- |
| **1.1 Check Git Installation** | git --version | git version 2.45.1.windows.1 *(version may vary)* |
| **1.2 Set Username** | git config --global user.name "Your Name" | *(No output)* |
| **1.3 Set Email** | git config --global user.email "your.email@example.com" | *(No output)* |
| **1.4 Verify Config** | git config --list | user.name=Your Nameuser.email=your.email@example.comcore.editor=notepad++ *(after editor setup)* |
| **2.1 Test Notepad++** | notepad++ | *(Opens Notepad++, no terminal output)* |
| **2.2 Create Alias** | alias np='notepad++' | *(No output)* |
| **2.3 Test Alias** | np | *(Opens Notepad++, no terminal output)* |
| **2.4 Set Editor** | git config --global core.editor "notepad++" | *(No output)* |
| **2.5 Verify Editor Config** | git config --global -e | *(Opens Notepad++ showing user name, email, and editor config)* |
| **3.1 Create Project Folder** | mkdir GitDemocd GitDemo | *(No output)* |
| **3.2 Initialize Repo** | git init | Initialized empty Git repository in C:/Users/YourUser/GitDemo/.git/ |
| **3.3 Show Hidden Files** | ls -a | . .. .git |
| **3.4 Create File** | notepad++ welcome.txt | *(Opens Notepad++, you save content and close)* |
| **3.5 List Files** | ls | welcome.txt |
| **3.6 View File Content** | cat welcome.txt | Welcome to Git Demo! |
| **3.7 Check Status** | git status | Shows *Untracked files:*welcome.txt |
| **3.8 Add File to Staging** | git add welcome.txt | *(No output)* |
| **3.9 Commit Changes** | git commit | *(Notepad++ opens — you add a commit message — save & close)*[master (root-commit) 1a2b3c4] Initial commit with welcome.txt1 file changed, 1 insertion(+)create mode 100644 welcome.txt |
| **3.10 Status After Commit** | git status | On branch masternothing to commit, working tree clean |
| **4.1 Add Remote Origin** | git remote add origin https://gitlab.com/yourusername/GitDemo.git | *(No output)* |
| **4.2 Pull Remote (Optional)** | git pull origin master | warning: no common commits *(if repo empty)*Already up to date. |
| **4.3 Push to Remote** | git push origin master | Enumerating objects: 3, done.Counting objects: 100% (3/3), done.Delta compression using up to 8 threadsCompressing objects: 100% (2/2), done.`Writing objects: 100% (3/3), 280 bytes |

**EXERCISE-2**

## **Objectives**

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

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

* Implement git ignore command to ignore unwanted files and folders

## **Prerequisites**

The following are the pre-requisites to complete this hands-on lab:

* Setting up Git environment
* Integrate notepad++ as a default editor
* A Git repository in the local system and a remote repository in GitLab

Notes\*:

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

Estimated time to complete this lab: **20 minutes.**

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.   
  
**COMMANDS AND OUTPUT:**

### **Step 1: Create a** .log **file and a** log **folder**

cd GitDemo

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

mkdir log

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

### **Step 2: Check status before ignoring**

git status

**Expected Output:**

Untracked files:

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

log/

test.log

### **Step 3: Create** .gitignore

notepad++ .gitignore

\*.log

log/

Save and close.

### **Step 4: Check status after adding** .gitignore

bash

Copy code

git status

**Expected Output:**

makefile

Untracked files:

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

.gitignore

Notice that test.log and log/ are **not** listed anymore — they are ignored.

### **Step 5: Add and commit** .gitignore

git add .gitignore

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

**Expected Output:**

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

1 file changed, 2 insertions(+)

create mode 100644 .gitignore

### **Step 6: Push to Remote**

git push origin master

**Expected Output:**

Enumerating objects: 3, done.

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

Delta compression using up to 8 threads

Compressing objects: 100% (2/2), done.

Writing objects: 100% (3/3), 250 bytes | 250.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0)

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

prevcommit..abc1234 master -> master

### **Step 7: Verify Ignoring Works**

* Create another .log file and check status:

echo "New log entry" > debug.log

git status

**Expected Output:**

On branch master

nothing to commit, working tree clean

## debug.log is ignored and not staged. **EXERCISE-3**

## **Objectives**

* Explain branching and merging
* Explain about creating a branch request in GitLab
* Explain about creating a merge request in GitLab

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

* Construct a branch, do some changes in the branch, and merge it with master (or trunk)

## **Prerequisites**

The following are the pre-requisites to complete this hands-on lab:

* Setting up Git environment with P4Merge tool for Windows

Notes\*:

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

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

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

**Branching:**

1. Create a new branch **“GitNewBranch”.**
2. List all the local and remote branches available in the current trunk. Observe the “\*” mark which denote the current pointing branch.
3. Switch to the newly created branch. Add some files to it with some contents.
4. Commit the changes to the branch.
5. Check the status with **“git status”** command.

**Merging:**

1. Switch to the master
2. List out all the differences between trunk and branch. These provide the differences in command line interface.
3. List out all the visual differences between master and branch using **P4Merge tool**.
4. Merge the source branch to the trunk.
5. Observe the logging after merging using **“git log –oneline –graph –decorate”**
6. Delete the branch after merging with the trunk and observe the git status.

**COMMANDS AND OUTPUTS:**

| **Step** | **Command** | **Output** |
| --- | --- | --- |
| **Branching — 1. Create New Branch** | git branch GitNewBranch | *(No output — branch created)* |
| **Branching — 2. List All Branches** | git branch -a | \* master GitNewBranch remotes/origin/master *(“*” shows current branch)\* |
| **Branching — 3. Switch to New Branch** | git checkout GitNewBranch | Switched to branch 'GitNewBranch' |
| **Branching — 4. Add File to Branch** | echo "Branch-specific content" > branchfile.txt | *(No output)* |
|  | git add branchfile.txt | *(No output)* |
| **Branching — 5. Commit Changes** | git commit -m "Add branch-specific file" | [GitNewBranch 1a2b3c4] Add branch-specific file 1 file changed, 1 insertion(+) create mode 100644 branchfile.txt |
| **Branching — 6. Check Status** | git status | On branch GitNewBranchnothing to commit, working tree clean |
| **Merging — 1. Switch to Master** | git checkout master | Switched to branch 'master'Your branch is up to date with 'origin/master'. |
| **Merging — 2. Show Differences (CLI)** | git diff master GitNewBranch | diff --git a/branchfile.txt b/branchfile.txtnew file mode 100644+Branch-specific content |
| **Merging — 3. Show Differences (P4Merge)** | git difftool master GitNewBranch | *(P4Merge window opens showing visual differences)* |
| **Merging — 4. Merge Branch into Master** | git merge GitNewBranch | Updating abc1234..1a2b3c4Fast-forward` branchfile.txt |
| **Merging — 5. View Merge Log** | git log --oneline --graph --decorate | \* 1a2b3c4 (HEAD -> master, GitNewBranch) Add branch-specific file\* abc1234 Initial commit |
| **Merging — 6. Delete Branch After Merge** | git branch -d GitNewBranch | Deleted branch GitNewBranch (was 1a2b3c4). |
| **Merging — 7. Final Status** | git status | On branch masternothing to commit, working tree clean |

## EXERCISE 4 **Objectives**

* Explain how to resolve the conflict during merge.

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

* 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.

## **Prerequisites**

The following are the pre-requisites to complete this hands-on lab:

* Hands-on ID: **“Git-T03-HOL\_001”**

Notes\*:

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

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

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

1. Verify if master is in clean state.
2. Create a branch **“GitWork”.** Add a file “hello.xml”.
3. Update the content of “hello.xml” and observe the status
4. Commit the changes to reflect in the branch
5. Switch to master.
6. Add a file **“hello.xml”** to the master and add some different content than previous.
7. Commit the changes to the master
8. Observe the log by executing **“git log –oneline –graph –decorate –all”**
9. Check the differences with Git diff tool
10. For better visualization, use P4Merge tool to list out all the differences between master and branch
11. Merge the bran to the master
12. Observe the git mark up.
13. Use 3-way merge tool to resolve the conflict
14. Commit the changes to the master, once done with conflict
15. Observe the git status and add backup file to the .gitignore file.
16. Commit the changes to the .gitignore
17. List out all the available branches
18. Delete the branch, which merge to master.
19. Observe the log by executing **“git log –oneline –graph –decorate”**

**COMMANDS AND OUTPUT:**

| **Step** | **Command** | **Output** |
| --- | --- | --- |
| **1. Verify master is clean** | git status | On branch masternothing to commit, working tree clean |
| **2. Create branch** | git branch GitWork | *(No output)* |
| **3. Switch to branch** | git checkout GitWork | Switched to branch 'GitWork' |
| **4. Add hello.xml in branch** | echo "<msg>Hello from branch</msg>" > hello.xml | *(No output)* |
|  | git add hello.xml | *(No output)* |
| **5. Commit branch file** | git commit -m "Add hello.xml in GitWork branch" | [GitWork 1a2b3c4] Add hello.xml in GitWork branch 1 file changed, 1 insertion(+) create mode 100644 hello.xml |
| **6. Switch to master** | git checkout master | Switched to branch 'master' |
| **7. Add different hello.xml in master** | echo "<msg>Hello from master</msg>" > hello.xml | *(No output)* |
|  | git add hello.xml | *(No output)* |
| **8. Commit master file** | git commit -m "Add hello.xml in master" | [master 2b3c4d5] Add hello.xml in master 1 file changed, 1 insertion(+) create mode 100644 hello.xml |
| **9. View log (all branches)** | git log --oneline --graph --decorate --all | \* 2b3c4d5 (HEAD -> master) Add hello.xml in master` |
| **10. Show diff (CLI)** | git diff master GitWork | Shows text difference: <msg>Hello from master</msg> vs <msg>Hello from branch</msg> |
| **11. Show diff (P4Merge)** | git difftool master GitWork | *(P4Merge opens showing visual differences)* |
| **12. Merge branch into master** | git merge GitWork | Auto-merging hello.xmlCONFLICT (content): Merge conflict in hello.xmlAutomatic merge failed; fix conflicts and then commit the result. |
| **13. View conflict markup** | cat hello.xml | < <<<<<<< HEAD`<br>`<msg>Hello from master</msg>`<br>`=======`<br>`<msg>Hello from branch</msg>`<br>`>>>>>>> GitWork` |
| **14. Resolve conflict (3-way merge)** | *(Open file in P4Merge or manually edit to final content)*git add hello.xml | *(No output)* |
| **15. Commit resolved file** | git commit -m "Resolve conflict in hello.xml" | [master 3c4d5e6] Resolve conflict in hello.xml |
| **16. Check status** | git status | On branch master`nothing to |

## **EXERCISE 5: Objectives**

* Explain how to clean up and push back to remote Git

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

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

## **Prerequisites**

The following are the pre-requisites to complete this hands-on lab:

* Hands-on ID: **“Git-T03-HOL\_002”**

Notes\*:

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

Estimated time to complete this lab: **10 minutes.**

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

1. Verify if master is in clean state.
2. List out all the available branches.
3. Pull the remote git repository to the master
4. Push the changes, which are pending from **“Git-T03-HOL\_002”** to the remote repository.
5. Observe if the changes are reflected in the remote repository.

**COMMANDS AND OUTPUT:**

| **Step No.** | **Instruction** | **Git Command(s)** | **Outcome** |
| --- | --- | --- | --- |
| 1 | Verify if master is in clean state | git status | Shows *“On branch master”* and *“nothing to commit, working tree clean”* if clean. |
| 2 | List out all the available branches | git branch -a | Lists all local and remote branches. |
| 3 | Pull the remote git repository to the master | git pull origin master | Updates local master with latest changes from remote. |
| 4 | Push pending changes from “Git-T03-HOL\_002” to the remote repository | git push origin master | Sends all local commits from master to the remote repository. |
| 5 | Observe if the changes are reflected in the remote repository | *(No Git command — check on GitHub/GitLab web interface)* | Confirms that all changes are visible in the remote repository. |