**Mandatory Exercises for GIT Week 8**  
  
**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



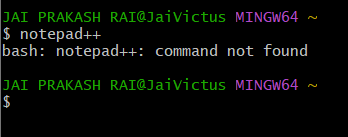
**welcome.txt** is added to the local repository.

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



Did all but I cant find the option to integrate notepad++ in my options!

**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.

**Git Ignore – Hands-on Lab Guide**

**Objectives**

You will be able to:

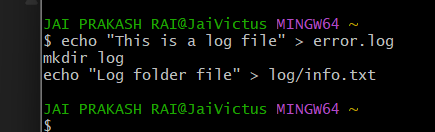
* Explain what .gitignore is.
* Ignore unwanted files and folders in a Git repository.

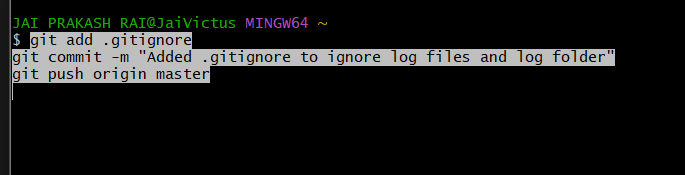
**1. Understanding .gitignore**

* .gitignore is a special text file that tells Git which files or directories to **ignore**.
* Ignored files are not tracked by Git, meaning they will not be staged, committed, or pushed to the remote repository.
* Common use cases:
  + Temporary files (.log, .tmp)
  + Build artifacts (/dist/, /build/)
  + OS-generated files (Thumbs.db, .DS\_Store)
  + Environment/config files with sensitive data

**2. How Git Ignore Works**

* Git reads patterns listed in the .gitignore file.
* If a file matches a pattern, Git **skips tracking** it.
* Ignoring only works for files **not already tracked** — if a file was committed before, you must remove it from tracking with:





**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.

**Theory**

**Branching in Git**

* Branching allows you to **work on a separate line of development** without affecting the main codebase (usually master or main).
* You can create as many branches as needed for different features, bug fixes, or experiments.
* Branches help in **parallel development**, where multiple people can work without interfering with each other.

**Merging in Git**

* Merging takes changes from one branch and **integrates them into another**.
* In most workflows, once a branch is tested and approved, it is merged into master or main.
* Merging can be done **fast-forward** (no conflicts) or with **merge commits** (if branches have diverged).

**Branch Request in GitLab**

* A **branch request** (also called creating a branch) in GitLab is simply adding a new branch in the repository either locally or via GitLab UI.
* You typically push your local branch to the GitLab remote repository to share it with your team.

**Merge Request in GitLab**

* A **merge request (MR)** in GitLab is a request to **merge changes from one branch into another**.
* It allows code review, discussion, and testing before the merge happens.

Creation:-  
  
# 1. Create a new branch called "GitNewBranch"

git branch GitNewBranch

# 2. List all branches (local and remote)

git branch -a

# 3. Switch to the new branch

git checkout GitNewBranch

# 4. Create a new file and add content

echo "This is content in the new branch" > branchfile.txt

# 5. Stage the file

git add branchfile.txt

# 6. Commit the changes

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

# 7. Check the status

git status

Merging:-  
# 1. Switch back to master

git checkout master

# 2. Show differences between master and GitNewBranch

git diff master GitNewBranch

# 3. Visual differences using P4Merge (ensure it's set up as diff tool)

git difftool master GitNewBranch

# 4. Merge GitNewBranch into master

git merge GitNewBranch

# 5. Show the commit history with a graph

git log --oneline --graph --decorate

# 6. Delete the branch after merging

git branch -d GitNewBranch

# 7. Check the status

git status  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**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”**

# 1. Verify if master is in clean state

git checkout master

git status

# 2. Create a branch “GitWork” and switch to it

git branch GitWork

git checkout GitWork

# 3. Add a file hello.xml with initial content

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

# 4. Stage and commit the file in GitWork

git add hello.xml

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

# 5. Switch back to master

git checkout master

# 6. Add a file hello.xml in master with different content

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

# 7. Stage and commit the file in master

git add hello.xml

git commit -m "Added hello.xml in master branch with different content"

# 8. View log with all branches

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

# 9. Check differences between master and GitWork

git diff master GitWork

# 10. View differences with P4Merge (if configured)

git difftool master GitWork

# 11. Attempt to merge GitWork into master (will cause conflict)

git merge GitWork

# 12. Check conflict markup

git status

# 13. Resolve conflict using 3-way merge tool (P4Merge example)

git mergetool

# 14. After resolving, stage and commit

git add hello.xml

git commit -m "Resolved merge conflict between master and GitWork"

# 15. Add backup files to .gitignore

echo "\*.orig" >> .gitignore

# 16. Stage and commit .gitignore changes

git add .gitignore

git commit -m "Added .gitignore entry for backup files"

# 17. List all branches

git branch

# 18. Delete the merged branch

git branch -d GitWork

# 19. View final log graph

git log --oneline --graph –decorate

**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.

# 1. Verify if master is in clean state

git checkout master

git status

# 2. List all available branches

git branch -a

# 3. Pull the latest changes from the remote repository into master

git pull origin master

# 4. Push the pending changes from “Git-T03-HOL\_002” to the remote repository

# (Replace <commit-message> with your actual commit message if not already committed)

git add .

git commit -m "Completed changes for Git-T03-HOL\_002"

git push origin master

# 5. Verify if changes are reflected in the remote repository

# (This step is usually done by checking GitHub/GitLab in a browser)

git log --oneline --graph --decorate –all  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
 *Thank You<>*