WEEK 8 GIT SOLUTION

**Ex.no:01**

**Step 1: Setup Git Configuration**

**Description:**  
Installed Git Bash and configured username and email.

**Commands Used:**

git --version

git config --global user.name " SwathiRajmohan"

git config --global user.email "swathiche0@gmail.com"

git config --global --list

**Step 2: Integrate Notepad++ with Git**

**Description:**  
Configured Notepad++ as the default Git editor.

**Commands Used:**

notepad++

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

git config --global -e

**Step 3: Create Local Repository**

**Description:**  
Created a folder GitDemo and initialized it as a Git repository.

**Commands Used:**

mkdir GitDemo

cd GitDemo

git init

**Step 4: Add File to Repository**

**Description:**  
Created welcome.txt file and committed it.

**Commands Used:**

echo "Welcome to Git" > welcome.txt

git add welcome.txt

git commit

**Step 5: Push to Remote Repository**

**Description:**  
Created GitLab repository and pushed local commits to it.

**Commands Used:**

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

git pull origin master

git push origin master

**Ex.no:02**

**Step 1: Setup Prerequisites**

**Description:**

* Git is installed and configured with username and email
* Default editor set to Notepad++
* Existing local Git repository linked to a GitLab remote repository

**Step 2: Create Sample Files to Ignore**

**Commands Used:**

echo "Temporary log data" > system.log

mkdir temp\_logs

echo "Server error details" > temp\_logs/server.txt

**Step 3: Create and Configure .gitignore**

**Commands Used:**

notepad++ .gitignore

**Content of .gitignore:**

\*.log

temp\_logs/

**Explanation:**

* \*.log → Ignores all files with .log extension
* temp\_logs/ → Ignores the entire temp\_logs folder

**Step 4: Check Ignored Files**

**Commands Used:**

git status

**Expected Output:**  
The .log file and temp\_logs folder should not appear in the untracked files list.

**Step 5: Add and Push .gitignore File**

**Commands Used:**

git add .gitignore

git commit -m "Configured .gitignore to exclude log files and temp\_logs folder"

git push origin master

**Ex.no:03**

**Step 1: Setup Prerequisites**  
**Description:**

* Git is already installed and configured with username and email
* P4Merge tool is available for visual merge comparisons
* Local Git repository linked with a remote GitLab repository

**Step 2: Branching**

**2.1 Create a New Branch**

git branch FeatureUpdate

**2.2 List All Branches**

git branch -a

* The branch with the \* symbol is the one you are currently on.

**2.3 Switch to the New Branch**

git checkout FeatureUpdate

**2.4 Add Files and Content**

echo "New functionality implemented" > update.txt

git add update.txt

**2.5 Commit Changes**

git commit -m "Added update.txt in FeatureUpdate branch"

**2.6 Check Branch Status**

git status

**Step 3: Merging**

**3.1 Switch to Master Branch**

git checkout master

**3.2 Compare Differences in CLI**

git diff master FeatureUpdate

**3.3 View Differences with P4Merge**

git mergetool

**3.4 Merge the Feature Branch into Master**

git merge FeatureUpdate

**3.5 Display Merge Log**

git log --oneline --graph --decorate

**3.6 Remove the Merged Branch**

git branch -d FeatureUpdate

**3.7 Verify Status After Merge**

git status

**Step 4: Creating a Branch Request in GitLab**

1. Push the branch to the remote repository:

git push origin FeatureUpdate

1. Navigate in GitLab: **Project → Repository → Branches**
2. Click **New Merge Request** to merge FeatureUpdate into master.

**Step 5: Creating a Merge Request in GitLab**

* Provide a title and short description for the merge request
* Assign reviewers if needed
* Click **Create Merge Request**
* Once approved, select **Merge**

**Ex.no : 04**

**Step 1: Environment Preparation**

**Details:**

* Git is fully set up and configured on the machine
* P4Merge is installed for visual conflict resolution
* Local repository is linked to the GitLab remote

**Step 2: Confirm Master Branch is Clean**

git checkout master

git status

* Expected message: *“working tree clean”*

**Step 3: Create Feature Branch and Add XML File**

git branch GitWork

git checkout GitWork

echo "<message>Branch says hello</message>" > hello.xml

git add hello.xml

**Step 4: Commit Changes to Branch**

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

**Step 5: Switch to Master and Make Different Update**

git checkout master

echo "<message>Master says hello</message>" > hello.xml

git add hello.xml

git commit -m "Created hello.xml in master with alternate text"

**Step 6: Review Commit History**

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

**Step 7: Compare Master and Branch**

**Using CLI:**

git diff master GitWork

**Using P4Merge:**

git mergetool

**Step 8: Merge and Cause Conflict**

git merge GitWork

* Git will indicate a conflict in hello.xml

**Step 9: Resolve Conflict via 3-Way Merge**

git mergetool

* In P4Merge, decide whether to keep changes from master, branch, or merge both

**Step 10: Commit After Resolving**

git add hello.xml

git commit -m "Merge conflict resolved in hello.xml"

**Step 11: Add Backup Files to .gitignore**

echo "\*.orig" >> .gitignore

git add .gitignore

git commit -m "Added .orig files to .gitignore"

**Step 12: Remove the Feature Branch**

git branch -d GitWork

**Step 13: Final Log Check**

git log --oneline --graph --decorate

**Ex.no:05**

**Step 1: Check Master Branch Status**

git checkout master

git status

* You should see: *“nothing to commit, working tree clean”* if the branch is up to date.

**Step 2: List Available Branches**

git branch -a

* The branch with the \* symbol is the one you are currently on.

**Step 3: Update Local Master from Remote**

git pull origin master

* This command fetches and merges the latest changes from the remote master branch.

**Step 4: Push Local Changes to Remote**  
If there are committed changes from **Git-T03-HOL\_002** that haven’t been uploaded yet:

git push origin master

**Step 5: Verify Changes on Remote Repository**

1. Go to your GitHub or GitLab project page.
2. Open the **Commits** or **Files** section.
3. Confirm that the most recent commits are showing in the remote repository.