1. Git Assignment: Ignoring .log Files and log Folder

**Steps to Perform the Task**

**Step 1: Create Files and Folders**

* Open your Git-tracked project directory (or initialize a new Git repository using git init).
* Create a new file with .log extension:

bash

touch debug.log

* Create a new folder named log:

bash

mkdir log

* (Optional) Add files inside the log folder:

bash

echo "Some content" > log/server.log

**Step 2: Update .gitignore File**

* Open (or create if it doesn’t exist) a file named .gitignore in the root of your project.
* Add the following lines:

bash

\*.log

log/

Explanation:

* \*.log — Ignores all files with .log extension.
* log/ — Ignores the entire log folder and its contents.

**Step 3: Verify with Git Status**

* Run the following command to check if the files are being ignored:

bash

git status

* Expected Output:

pgsql

On branch main

No changes added to commit but untracked files present (use "git add" to track)

.gitignore

1. Git Hands-on Lab: Branching & Merging

**Part 1: Branching**

**1. Create a new branch named GitNewBranch:**

**bash**

**git branch GitNewBranch**

**2. List all local and remote branches:**

**bash**

**git branch -a**

**The \* indicates the currently checked-out branch.**

**3. Switch to the newly created branch:**

**bash**

**git checkout GitNewBranch**

**4. Add some files and content to this branch:**

**bash**

**echo "This is a new file in GitNewBranch." > newfile.txt**

**git add newfile.txt**

**5. Commit the changes:**

**bash**

**git commit -m "Added newfile.txt in GitNewBranch"**

**6. Check the current status:**

**Bash**

**git status**

**You should see a clean working directory after commit.**

**Part 2: Merging**

**1. Switch back to the master (or main) branch:**

**bash**

**git checkout master**

**2. List the differences between master and GitNewBranch:**

**bash**

**git diff master..GitNewBranch**

**3. Visual diff using P4Merge (if installed):**

**bash**

**git difftool master..GitNewBranch**

***Ensure P4Merge is configured as your Git difftool.***

**4. Merge GitNewBranch into master:**

**bash**

**git merge GitNewBranch**

**This integrates the changes made in the feature branch into the master branch.**

**5. View the merge log visually:**

**bash**

**git log --oneline --graph --decorate**

**This shows a compact graphical representation of branch history.**

**6. Delete the merged branch:**

**bash**

**git branch -d GitNewBranch**

**7. Confirm everything is up to date:**

**Bash**

**git status**

**Output should say: *"On branch master. nothing to commit, working tree clean."***

1. **Git Hands-on Lab: Conflict Resolution & Merge**
2. **Verify if master is in clean state**

**bash**

**git checkout master**

**git status**

**2. Create a new branch GitWork and add hello.xml**

**bash**

**git checkout -b GitWork**

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

**git add hello.xml**

**3. Update the content of hello.xml and observe status**

**bash**

**echo "<message>Updated content in GitWork branch</message>" > hello.xml**

**git status**

**4. Commit the changes in GitWork**

**bash**

**git add hello.xml**

**git commit -m "Updated hello.xml in GitWork branch"**

**5. Switch to master**

**bash**

**git checkout master**

**6. Add a different hello.xml in master**

**bash**

**echo "<message>Different content from master branch</message>" > hello.xml**

**git add hello.xml**

**7. Commit the changes in master**

**bash**

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

**8. Observe the log graphically**

**bash**

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

**9. Check textual differences**

**bash**

**git diff GitWork**

**10. Use P4Merge (visual tool) to see differences**

**bash**

**git difftool master GitWork**

***(Make sure P4Merge is configured as the difftool beforehand)***

**11. Attempt to merge GitWork into master**

**bash**

**git merge GitWork**

**Expect a merge conflict in hello.xml.**

**12. Observe the Git markup (conflict markers)**

**bash**

**cat hello.xml**

**Look for:**

**markdown**

**<<<<<<< HEAD**

**(Different content from master)**

**=======**

**(Updated content in GitWork)**

**>>>>>>> GitWork**

**13. Use 3-way merge tool (like VSCode or P4Merge)**

**bash**

**git mergetool**

**Select appropriate version or manually edit to resolve.**

**14. Commit resolved conflict**

**bash**

**git add hello.xml**

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

**15. Check status & ignore backup file**

**bash**

**git status**

**echo "\*.orig" >> .gitignore**

**16. Commit the updated .gitignore**

**bash**

**git add .gitignore**

**git commit -m "Ignored backup files using .gitignore"**

**17. List all branches**

**bash**

**git branch**

**18. Delete the merged GitWork branch**

**bash**

**git branch -d GitWork**

**19. Final log check**

**Bash**

**git log --oneline --graph –decorate**

1. **Git Hands-on Lab: Remote Sync & Branch Push**

**1. Verify if master is in a clean state**

**bash**

**git checkout master**

**git status**

**Make sure there are no uncommitted changes.**

**2. List out all available branches**

**bash**

**git branch -a**

**This shows local (master, Git-T03-HOL\_002, etc.) and remote branches (remotes/origin/...).**

**3. Pull the remote git repository into master**

**bash**

**git pull origin master**

**Ensures the latest remote changes are synced locally.**

**4. Push pending changes from Git-T03-HOL\_002 to remote**

**bash**

**git checkout Git-T03-HOL\_002**

**git push origin Git-T03-HOL\_002**

**This sends your local branch commits to the remote Git server.**

**5. Observe if changes are reflected remotely**

**You can verify in two ways:**

**A. From command line:**

**bash**

**git log origin/Git-T03-HOL\_002 --oneline**

**B. Or visually:**

* **Open your repository on GitHub / GitLab / Bitbucket**
* **Check under the Branches or Commits section**