## **WEEK-8**

**3. Git – HOL**

* Explain branching and merging

Branching in Git allows you to create a separate line of development from the main project. This is useful when you want to work on a new feature or fix a bug without affecting the main code. Merging is the process of combining the changes from one branch into another, usually merging the feature branch back into the main branch once the work is complete.

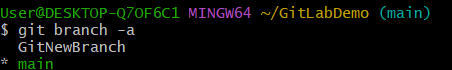
* Explain about creating a branch request in GitLab

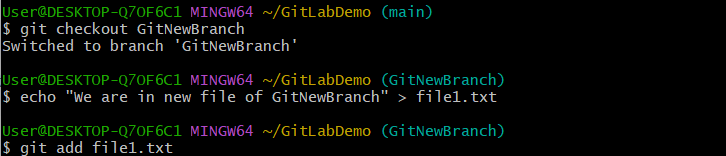
In GitLab, a branch request usually means creating a new branch to work on specific changes. You can do this directly in the GitLab repository by going to the Repository → Branches section and clicking “New Branch”. You give it a name (like feature-login) and choose the branch from which it should be created (often main or master).

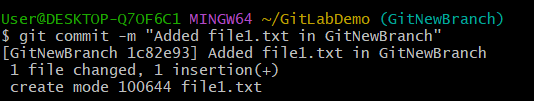
* Explain about creating a merge request in GitLab

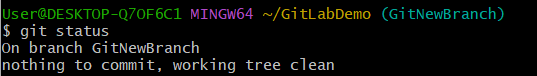
A merge request in GitLab is a way to ask for your changes in one branch to be merged into another branch. It also gives other team members a chance to review your code before merging. You can create a merge request by going to Merge Requests → New Merge Request, selecting the source branch (your feature branch) and the target branch (usually main), and then submitting it for review.



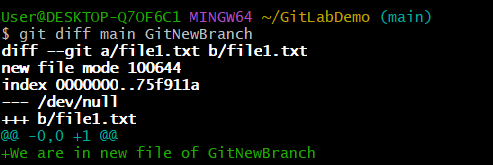




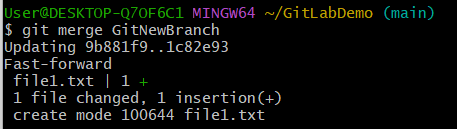


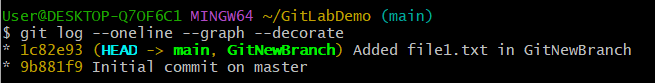


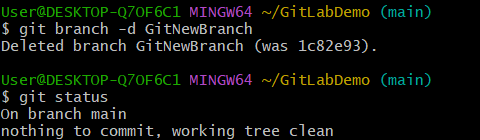












Explanation:

* Branching allows parallel work – In Git, creating a branch lets you work on new features or changes without disturbing the main code. It’s like having a separate copy of the project where you can experiment safely.
* Creating the branch – I created a new branch named GitNewBranch using the git branch GitNewBranch command. This branch was separate from the main (master) branch.
* Checking available branches – Using git branch -a, I listed all local and remote branches. The branch with a \* symbol was the one I was currently working on.
* Switching to the new branch – I moved into my new branch with git checkout GitNewBranch. This allowed me to start making changes in it without affecting the main branch.
* Adding and committing changes – I created new files, added content, then staged them using git add <filename> and committed them with git commit -m "Added new files in GitNewBranch".
* Checking status – I used git status to confirm that all changes were committed and my branch was up to date.
* Switching back to master – To prepare for merging, I switched to the master branch using git checkout master.
* Comparing branches – I compared master and GitNewBranch in two ways:
* Command line differences using git diff master GitNewBranch
* Visual differences using the P4Merge tool, which displayed changes in a graphical format.
* Merging the branch – I merged my work from GitNewBranch into master using git merge GitNewBranch. After merging, git log --oneline --graph --decorate showed a clear history of the merge.
* Deleting the branch – Since the changes were now in master, I deleted the branch with git branch -d GitNewBranch. Running git status confirmed I was back on master with no unmerged changes.