**2.Git-HOL**

**Q1: Explain git ignore**

**Answer:**  
Git ignore is a feature that allows you to tell Git which files or folders it should not track or include in your repository. This is done by using a special file named .gitignore, where you list file names or patterns that you want Git to ignore. It helps prevent unwanted files like logs, temporary files, or build outputs from being added to the repository.

**Q2: Explain how to ignore unwanted files using git ignore**

**Answer:**  
To ignore unwanted files, create a .gitignore file in your project directory and add the names or patterns of the files/folders you want Git to ignore. For example, adding these lines to .gitignore will ignore all .log files and the temp folder:

pgsql

\*.log

temp/

Save the file, and Git will stop tracking any files that match those patterns when you run commands like git add or git commit.

**3.Git-HOL**

**Q1: Explain branching and merging**

**Answer:**  
Branching in Git allows you to create a separate line of development to work on features or fixes without affecting the main code. Merging is the process of combining the changes from one branch back into another, typically merging a feature branch into the main branch once the work is done.

**Q2: Explain about creating a branch request in GitLab**

**Answer:**  
Creating a branch request in GitLab means creating a new branch in the repository. You do this by going to **Repository > Branches** in GitLab, clicking **New branch**, entering the branch name, selecting the base branch, and clicking **Create branch**.

**Q3: Explain about creating a merge request in GitLab**

**Answer:**  
A merge request in GitLab is a way to propose merging changes from one branch into another. To create one, push your changes to a branch, go to **Merge Requests** in GitLab, click **New merge request**, select the source and target branches, add details like title and description, and submit it for review and approval.

**4.Git-HOL**

**Q1: Explain how to resolve the conflict during merge.**

**Answer:**  
A merge conflict occurs when Git cannot automatically combine changes from two branches because the same part of a file was changed differently. To resolve it:

1. Check which files have conflicts after the merge.
2. Open the conflicted files and find conflict markers (<<<<<<<, =======, >>>>>>>).
3. Manually edit the file to keep the desired changes and remove the conflict markers.
4. Stage the resolved files using git add.
5. Complete the merge by committing with git commit.

This process resolves conflicts and finalizes the merge.

**5.Git-HOL**

**Q1: Explain how to clean up and push back to remote Git.**

**Answer:**  
Cleaning up and pushing back to remote Git involves preparing your local repository before sending your changes to the remote repository. Here are the steps:

**Clean up your working directory:**

Remove untracked files or directories using:

bash

git clean -f # for files

git clean -fd # for files and directories

Remove or stash any local changes you don’t want to commit:

bash

git stash # temporarily saves changes

git stash pop # restores stashed changes

**Stage your changes:**

bash

git add .

**Commit your changes with a meaningful message:**

bash

git commit -m "Your commit message"

**Push changes to the remote repository:**

bash

git push origin branch-name

This sends your local commits to the remote repository, keeping it updated.