TASK 1- DevOps

Question 1: Your organization uses GitHub and Bitbucket to store code repositories. You have cloned a repository onto your local system and later changed the directory name. After some days, a team member asks you to share the clone link of the repository. How would you provide the same?

Answer: When you clone a repository from GitHub or Bitbucket, the remote URL remains the same regardless of local directory name changes. To find the correct repository link, follow these steps:

1. Check the Remote URL in Your Local Repository

If already cloned the repository, retrieve its original remote URL using the following command:

git remote-v

Example Output:

origin https://github.com/your-org/repo-name.git (fetch)
origin https://github.com/your-org/repo-name.git (push)

This command lists all configured remote repositories.

2. Share the Clone URL

Based on the output of git remote-v, can share either the **HTTPS** or **SSH** URL with team member

For HTTPS:

https://github.com/your-org/repo-name.git

For SSH:

git@github.com:your-org/repo-name.git

3. Get the Clone URL from GitHub or Bitbucket UI

Alternatively,can find the clone URL directly from the web interface:

- **GitHub:**

1. Navigate to the repository in GitHub.
2. Click on the "Code" button.
3. Copy the HTTPS or SSH URL displayed.

- **Bitbucket:**

1. Open the repository in Bitbucket.
2. Click on the "Clone" option.
3. Copy the repository URL.

4. Verify If the Repository is Up-to-Date

Before sharing the repository, ensure that local copy is up-to-date with the remote repository:

git fetch--all

This will pull the latest changes and confirm that the repository is still accessible.

Conclusion:

Even if you rename the local directory, the remote URL remains unchanged. You can always retrieve it using git remote-v or check GitHub/Bitbucket UI to share the correct clone link with your team.

Question 2: I have a shell script to delete a particular dependency (repo is maven project). Before running the script I need to clone repo to my local, here point to note i should only clone master branch and only last commit (last commit has all the code) how would you do this?

Solution: To accomplish this, a **shallow clone** of the repository is required. This ensures that only the latest commit of the master branch is cloned, minimizing the download size and improving efficiency.

Implementation:

The following Git command is used:

git clone--depth 1--single-branch--branch master <repo_URL>

Step-by-Step Execution:

1. **Open a terminal** and navigate to the desired directory.
2. **Execute the command:**

git clone--depth 1--single-branch--branch master https://github.com/organization/repository.git

- --depth 1: Clones only the latest commit.
- --single-branch--branch master: Clones only the master branch.

3. **Change into the cloned repository directory:**

   cd repository

Run the shell script to delete the required dependency.

Advantages of Shallow Cloning:

- **Reduces data transfer** by downloading only the latest commit.
- **Faster cloning process**, especially for large repositories.
- **Avoids unnecessary commit history**, keeping the repository lightweight.

Conclusion:

Using the--depth 1--single-branch--branch master option enables efficient cloning of only the latest version of the master branch, ensuring an optimized setup before executing the shell script.

Question 3 What is a submodule in Git, and why is it needed?

Solution: A submodule in Git is a repository embedded inside another repository. It allows one repository to include another repository as a dependency while keeping both repositories separate.

Why Use a Git Submodule?

- Modular Code Management: Helps manage separate codebases while integrating them into a larger project.
- Version Control for Dependencies: Ensures that a project always uses a specific version of a dependency.
- Avoids Code Duplication: Instead of copying and pasting code, it links directly to an external repository.
- Independent Development: Allows working on submodules separately without affecting the main repository.

How to Add a Submodule?

To add a submodule to a repository, use:

git submodule add <repo_URL> <submodule_path>

Example:

git submodule add https://github.com/example/library.git external-library

Working with Submodules

- Initialize Submodules (After Cloning a Repo with Submodules):

  git submodule init
  git submodule update

- Update Submodules to the Latest Commit:

  git submodule update--remote

- Remove a Submodule:

  git submodule deinit-f <submodule_path>
  rm-rf .git/modules/<submodule_path>
  git rm-f <submodule_path>

Conclusion:

Git submodules provide a way to manage external repositories as part of a project without merging their history into the main repository. They are useful for managing dependencies, shared code, or modular projects while keeping repositories independent.

Question 4 : In a Git repository, five files (a, b, c, d, and e) have been modified and staged using git add .. Now, the requirement is to remove file d from the staging area without affecting other staged files. How can this be achieved?

Solution: To remove a specific file from the staging area without deleting the actual file, use the following command:

git reset HEAD <file_name>

Implementation:

1. Check the status of staged files:

   git status

   This will list all the files that are staged for commit.

2. Remove file d from the staging area:

   git reset HEAD d

   This command un-stages the file but keeps the changes in the working directory.

3. Verify the file is unstaged:

   git status

   File d should now be listed under "Changes not staged for commit."

4. Proceed with committing the remaining files:

   git commit-m "Committing files a, b, c, and e"

Alternative Approach (Using Git Restore- For Git 2.23+)

If using Git version 2.23 or later, the following command can be used instead:

git restore--staged d

This achieves the same result as git reset HEAD d.

Conclusion:

By using git reset HEAD d or git restore--staged d, file d is removed from the staging area while retaining its changes in the working directory. The remaining files can then be committed without including file d.