Git and GitHub Test Questions(05-07-2024) Haritha P V

1.What is Git and why is it used?

Git is a distributed version control system for efficiently handling projects of all sizes, widely used for source code management in software development. It ensures fast performance and robust collaboration features like branching and merging.
Version Control: Git tracks changes to files, allowing multiple developers to collaborate efficiently and revert to previous states when necessary.
Branching and Merging: Git supports branches, enabling developers to work on different features or fixes simultaneously. Merging allows these changes to be integrated back into the main codebase.
Distributed System: Every developer has a full copy of the project history locally, which allows for offline work and greater redundancy.
Data Integrity: Git ensures the integrity of the source code, preventing corruption and ensuring reliable change tracking.
Performance: Git is optimized for speed, handling large projects and complex histories with efficiency, making it suitable for both small and large projects.

2.Explain the difference between Git pull and Git fetch.

Git Pull

- Fetches changes from the remote repository and directly merges them into your current branch.
- Combines two commands, `git fetch` followed by `git merge`.
- Useful when you want to update your local branch with changes from the remote branch in one step.
- Can lead to conflicts that need to be resolved immediately as it merges changes automatically.

Git Fetch

- Downloads changes from the remote repository but does not merge them into your current branch. It updates the remote tracking branches.

- Only updates the remote tracking branches and leaves your working directory unchanged.
- Useful for checking what others have committed to the repository without affecting your working branch. This allows you to review the changes before merging.
- Safer for inspecting changes first and deciding when and how to merge them, reducing the risk of conflicts.

3.How do you revert a commit in Git?

git revert is a command in Git that creates a new commit which undoes the changes introduced by a specified previous commit.
Unlike git reset, which can alter the commit history, git revert maintains a clear and complete history of all changes, including those that have been undone.

```
git revert HEAD~3
```

Revert the changes specified by the fourth last commit in HEAD and create a new commit with the reverted changes

```
git revert -n master~5..master~2
```

Revert the changes done by commits from the fifth last commit in master (included) to the third last commit in master (included), but do not create any commit with the reverted changes. The revert only modifies the working tree and the index.

4.Describe the Git staging area.

The Git staging area, also known as the "index," is a critical part of the Git workflow. It acts as an intermediate area where changes are gathered before they are committed to the repository.

#Use git add to stage the changes you want to include in the next commit git add filename.txt

#to stage all changes, you can use:

git add .

#Once you are satisfied with the staged changes, commit them to the repository.

git reset HEAD filename.txt

#View staged changes

git status

5.What is a merge conflict, and how can it be resolved?

- A merge conflict occurs when Git cannot automatically resolve differences between two commits. This usually happens when changes are made to the same lines in a file or when one branch deletes a file that another branch modifies.
- During a git merge, if a conflict arises, Git will mark the conflicted areas in the affected files and pause the merge process. The git status command will list the files with conflicts.
- Git inserts conflict markers in the files where it detects conflicts.

  <<<<<<< HEAD Change from current branch ======= Change from merging branch >>>>>>> branch-name

- Once conflicts are resolved, stage the changes using `git add`.

  git add conflicted-file.txt

- Complete the merge with `git commit`.

6.How does Git branching contribute to collaboration?

- Isolated Development: Branches allow developers to work independently on different features or fixes.
- Parallel Workflows: Enable multiple developers to work concurrently without conflicts.
- Code Review: Facilitate thorough code reviews and discussions through pull requests.
- Safe Experimentation: Allow for experimentation without affecting the main codebase.
- Simplified Integration: Make it easier to integrate and manage changes from different branches.

7.What is the purpose of Git rebase?

The purpose of Git rebase is to integrate changes from one branch into another by applying each commit of the current branch on top of the target branch. It essentially rewrites the commit history to ensure a linear sequence of commits.

History Simplification: Git rebase helps maintain a linear commit history by integrating changes from one branch onto another.

Conflict Resolution: It allows conflicts to be addressed immediately as each commit is applied, rather than at the end of a merge process.

Interactive Rebase: Enables interactive editing of commit history, including squashing, reordering, or editing commits before integration.

Integration Preparation: Ensures that changes from a feature branch are up-to-date with the latest changes in the target branch, facilitating smoother merges.

8. Explain the difference between Git clone and Git fork.

Git Clone: Copies a repository to your local machine for development or collaboration.

Git Fork: Creates a copy of a repository on GitHub under your account, enabling independent experimentation and contribution to the original repository through pull requests.

9.How do you delete a branch in Git?

Local Branch: Use `git branch -d <branch-name>` to delete a merged local branch, or git branch -D <branch-name> to force delete an unmerged branch.

Remote Branch: Use `git push origin --delete <branch-name>` to delete a branch from the remote repository.

10.What is a Git hook, and how can it be used?

Git hooks are scripts that Git executes before or after events such as commits, merges, and pushes. They are located in the '.git/hooks' directory of every Git repository.

Benefits:

- **Automation**: Reduces manual tasks and ensures consistent processes.
- **Quality Assurance**: Enforces code quality standards and prevents erroneous commits.
- **Integration**: Seamlessly integrates with CI/CD pipelines for automated testing and deployment.
- **Customization**: Allows customization based on project-specific requirements and policies.

To create a Git hook, add an executable script with a specific filename (e.g., pre-commit, post-commit) in the .git/hooks directory of your Git repository.

To enforce code formatting,you need to specify that in the first line of the script

```
#!/bin/bash

# Example: Run tests before allowing a commit

 pytest

# Example: Check for linting issues

 pylint .

# Ensure the script exits with a non-zero status to abort the commit if necessary
exit 0
```