**Git & GitHub (Q&A)**

1. **What is Git and why is it used?**

Git is a distributed version control system used to track changes in source code during software development. It allows multiple developers to work on a project simultaneously without interfering with each other's changes. Git manages and records the history of changes, making it easier to revert to previous states, collaborate on code, and manage different versions of a project.

1. **Explain the difference between Git pull and Git fetch.**

Git pull: This command fetches the changes from a remote repository and merges them into the current branch in your local repository. Essentially, it is a combination of git fetch followed by git merge.

Git fetch: This command retrieves the latest changes from a remote repository but does not merge them into working directory. It updates your local copy of the remote branches, allowing to review the changes before deciding to merge them.

1. **How do you revert a commit in Git?**

To revert a commit, we can use the `git revert <commit-hash>` command. This creates a new commit that undoes the changes made by the specified commit, preserving the history of changes. Alternatively, we can use `git reset <commit-hash>` to reset the branch to a previous state.

1. **Describe the Git staging area.**

The Git staging area (or index) is an intermediate area where changes are listed before they are committed to the repository. When we modify files in our working directory, we use the `git add <file>` command to add those changes to the staging area. This allows us to prepare and review changes before finalizing them with a commit.

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

A merge conflict occurs when Git is unable to automatically reconcile differences between two commits that are being merged. This usually happens when changes are made to the same lines of a file or when one file is edited in different ways in separate branches. To resolve a merge conflict, we need to manually edit the conflicting files to reconcile the differences and then mark them as resolved with `git add <file>`. Finally, complete the merge with `git commit`.

1. **How does Git branching contribute to collaboration?**

Git branching allows multiple developers to work on different features, bug fixes, or experiments in parallel without affecting the main codebase. Each branch can be developed independently, and changes can be merged back into the main branch when ready. This enables efficient collaboration and reduces the risk of conflicts or disruptions in the main codebase.

1. **What is the purpose of Git rebase?**

Git rebase is used to integrate changes from one branch into another by moving or combining a sequence of commits. It allows us to maintain a cleaner, linear project history by transferring the completed work onto the tip of another branch. Unlike merging, rebasing replays the commits on top of the target branch, which can make the history easier to read but can also rewrite commit history, potentially causing issues in shared repositories.

1. **Explain the difference between Git clone and Git fork.**

Git clone: This command creates a local copy of a remote repository, allowing us to work on the project on local machine. It is typically used to start working on an existing project.

Git fork: This is a copy of a repository that resides on our own GitHub (or similar service) account. It is used to propose changes to someone else's project. Forking is often the first step in contributing to an open-source project, as it allows us to freely experiment with changes without affecting the original repository.

1. **How do you delete a branch in Git?**

To delete a branch in Git, the following commands are used:

Locally: `git branch -d <branch-name>` (or -D to force delete).

Remotely: `git push origin --delete <branch-name>`.

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

Git hooks are custom scripts that are triggered by various Git commands and actions. They allow us to automate tasks and enforce policies. There are two types of hooks: client-side (triggered by operations such as commit, merge, and push) and server-side (triggered by network operations like receiving pushed commits). For example, a pre-commit hook can be used to run tests or linters before allowing a commit, ensuring code quality and consistency.