Git Interview Q&A

Question-1: What happens when you run git fetch?

Answer: git fetch downloads new changes from a remote repository, but does not apply them to your local branch.

It updates your remote-tracking branches so that they match the state of the corresponding branches on the remote repository.

Question-2: What is the difference between git merge and git rebase?

Answer: git merge combines changes from one branch into another branch, creating a new merge commit that has two parent commits.

git rebase, on the other hand, rewrites the history of a branch by moving the branch to the tip of another branch and applying the changes in a linear fashion.

This creates a cleaner, easier-to-understand commit history, but can lead to conflicts if multiple people are working on the same branch.

Question-3: You have made a commit that contains sensitive information, such as a password, and you want to remove it from the Git repository. What is the best way to do this?

Answer: The best way to remove a commit that contains sensitive information from the Git repository is to use the git filter-branch command.

First, use the git log command to identify the commit that contains the sensitive information.

Then, use the git filter-branch --force --index-filter 'git rm --cached --ignore-unmatch path/to/file'

-- prune-empty --tag-name-filter cat- -all command to remove the sensitive information from the commit and all subsequent commits.

Finally, use the git push --force command to update the remote repository.

Question-4: How do you resolve conflicts in Git?

Answer: To resolve conflicts in Git, you first need to identify which files have conflicts by running git status.

Then, open the conflicted files in a text editor and look for the sections marked with <<<<<<<, =======, and >>>>>>>.

Edit the files to remove the conflict markers and make the necessary changes to resolve the conflicts.

Once you have resolved all conflicts, stage the changes with git add and commit the changes with git commit.

Question-5: What is the difference between git push and git push –force?

Answer: git push sends your local changes to a remote repository and updates the remote branches to match your local branches. git push --force, however, overwrites the remote branches with your local branches, even if they have diverged.

This can cause you to lose changes if you are not careful, so it should be used with caution.

Question-6: What is the difference between git reset and git revert?

Answer: git reset removes commits from the current branch, moving the branch pointer back to an earlier commit.

This can be useful if you want to undo local changes that have not been pushed to a remote repository. git revert, on the other hand, creates a new commit that undoes the changes made by a previous commit.

This is useful if you want to undo changes that have already been pushed to a remote repository and shared with others.

Question-7: You have made a mistake in a commit message and want to change it. What is the best way to do this?

Answer: The best way to change a commit message in Git is to use the git commit --amend command.

First, use the git log command to identify the commit with the incorrect message. Then, use the git commit --amend -m "corrected message" command to change the commit message.

Finally, use the git push --force command to update the remote repository.

Question-8: What is Git reflow?

Answer: Git reflow is a tool that simplifies Git workflows by automating the creation of pull requests and other common Git tasks.

It provides a set of commands that allow developers to focus on writing code, rather than managing Git branches and pull requests.

Question-9: What is the difference between Git and GitHub?

Answer: Git is a distributed version control system that allows developers to track changes in source code over time.

GitHub, on the other hand, is a web-based platform that provides hosting for Git repositories, as well as additional features such as issue tracking, code review, and collaboration tools.

Question-10: What is a Git hook?

Answer: A Git hook is a script that runs automatically before or after a Git command is executed.

Hooks can be used to perform tasks such as checking for code quality, enforcing commit message formatting, and sending notifications.

Git provides several pre-defined hooks, and developers can also create their own custom hooks.

Question-11: You have accidentally deleted a file that was committed to your Git repository. What is the best way to restore the file?

Answer: The best way to restore a deleted file in Git is to use the git checkout

command. First, identify the commit in which the file was last present using the git log

command.

Then, use the git checkout <commit> -- <file> command to restore the file from the specified commit.

If the file was deleted in the most recent commit, you can use the git reset HEAD~1 command to reset the branch to the previous commit, use git checkout to restore the file, and then use the git add and git commit commands to commit the restored file.

Question-12: What is Git bisect?

Answer: Git bisect is a tool that helps you find the commit that introduced a bug. It works by doing a binary search of the commit history, asking you to test a commit in the middle of the range each time.

Based on your feedback, it will narrow down the range of possible commits until it finds the one that introduced the bug.

Question-13: What is a Git stash?

Answer: A Git stash is a way to temporarily save changes that you are not ready to commit, so that you can work on a different branch or pull in changes from another repository.

Stashing your changes creates a new "stash" object that stores your changes, and you can apply the stash later using git stash apply or git stash pop.

Question-14: You have made a commit to a branch, but you want to move it to a different branch. What is the best way to do this?

Answer: The best way to move a commit to a different branch in Git is to use the git cherry-pick command.

First, identify the commit hash using the git log command. Then, switch to the target branch using the git checkout command.

Finally, use the git cherry-pick <commit> command to apply the commit to the target branch.

This will create a new commit with the changes from the original commit applied to the target branch.

Question-15: What is Git LFS?

Answer: Git LFS (Large File Storage) is an extension for Git that allows you to manage large files such as audio, video, and graphics, that Git cannot handle efficiently.

Instead of storing the large files in the Git repository, Git LFS stores a pointer to the large file in the repository, and the actual file is stored in a separate Git LFS server.

Question-16: What is a Git submodule?

Answer: A Git submodule is a way to include a separate Git repository as a subdirectory of another repository.

This is useful when you want to include a library or other code from a separate repository, without copying the code into your repository.

The submodule is stored as a pointer to a specific commit in the separate repository, so that other developers can update the submodule to the latest version of the code.

Question-17: What is Git cherry-pick?

Answer: Git cherry-pick is a command that allows you to apply a specific commit from one branch to another branch.

This is useful when you want to apply a bugfix or feature from one branch to another branch without merging the entire branch.

The cherry-picked commit is treated as a new commit, so it will have a new commit ID.

Question-18: You have made some changes to a file, but you want to see what the file looked like before the changes. What is the best way to view the previous version of the file?

Answer: The best way to view the previous version of a file in Git is to use the git show command.

First, use the git log command to identify the commit in which the file was last modified.

Then, use the git show <commit>:path/to/file command to view the contents of the file at the specified commit.

If the file has been modified multiple times, you can use the git log --oneline path/to/file command to identify the commit hash for the previous version of the file, and then use the git show <commit> command to view the contents of the file at the specified commit.

Question-19: What is Git reflog?

Answer: Git reflog is a log of all the changes to Git references (such as branches, tags, and HEAD) that have occurred in a repository.

It can be used to recover lost commits, branches, or tags that have been deleted or overwritten.

The reflog can be accessed using the git reflog command.

Question-20: What is Git filter-branch?

Answer: Git filter-branch is a command that allows you to rewrite the history of a Git repository.

It can be used to remove sensitive data (such as passwords or API keys) from the commit history, or to split a large repository into smaller repositories.

The filter-branch command works by applying a set of filters to each commit in the repository, and creating new commits with the filtered changes.