Interview Questions

1. What is Version Control?

Answer: Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. It allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, and more.

2. What is Git and why is it used?

Answer: Git is a distributed version control system that tracks changes to a source code. It allows multiple developers to work on a project simultaneously, without overwriting each other's changes. Git also keeps a complete history of all changes.

3. What is a repository in Git?

Answer: A repository in Git is a directory for your project. It contains all the project's files and stores each file's revision history. Repositories can have multiple collaborators and can be either public or private.

4. What is the command to clone a repository in Git?

Answer: The command to clone a repository in Git is `git clone <repository-url>`. This command creates a local copy of the repository on your system.

5. How do you stage and commit changes in Git?

Answer: To stage changes, you use the command `git add <filename>` or `git add .` to stage all changes. To commit these staged changes, you use the command `git commit -m "Commit message"`.

6. What is the difference between 'git pull' and 'git fetch'?

Answer: `git pull` does a `git fetch` followed by a `git merge`. In other words, `git pull` retrieves the latest changes from the remote repository and merges them into your local branch. On the other hand, `git fetch` only retrieves the latest changes, but doesn't merge them.

7. What is a branch in Git?

Answer: A branch in Git is an independent line of development. You can use branches to isolate work when you're developing features or fixes. The default branch in Git is the **main** branch.

8. How do you create and switch to a new branch in Git?

Answer: To create a new branch, you use the command `git branch <branch-name>`. To switch to this branch, you use the command `git checkout <branch-name>`.

9. What is a merge conflict in Git and how can it be resolved?

Answer: A merge conflict occurs when Git is unable to automatically resolve differences in code between two commits. When a conflict happens, Git will pause the merge and ask you to resolve the conflict. You resolve the conflict by manually editing the conflicted file, then you add the resolved file to staging using `git add <filename>`, and commit it using `git commit`.

10. What is GitHub and how is it related to Git?

Answer: GitHub is a web-based hosting service for Git repositories. It provides a web-based graphical interface and access control, along with several collaboration features like repositories, branches, commits, Pull Requests, and more.

11. What is a Pull Request on GitHub?

Answer: A Pull Request is a feature on GitHub where a developer can contribute to a project by making changes to the project's repository and then asking the project's owner to review and pull in their contribution.

12. How do you handle a large Pull Request on GitHub?

Answer: Large Pull Requests can be difficult to review. It's better to split them into smaller, logical chunks. If a large Pull Request is unavoidable, you can use GitHub's review feature to comment on specific lines or sections, and use the 'Files changed' tab to get an overview of changes.

13. What are Git hooks?

Answer: Git hooks are scripts that Git executes before or after events such as commit, push, and receive. They are used for automating tasks in the development workflow, like running tests before a commit.

14. What is Git rebase and how is it different from merge?

Answer: Git rebase is a command that allows you to move or combine a sequence of commits to a new base commit. It's different from merge, as merging takes the contents of a source branch and integrates it with the target branch, while rebasing moves or combines the commits to the target branch.

15. What is a Git Stash?

Answer: Git stash takes your modified tracked files and saves them on a stack of unfinished changes that you can reapply at any time. It's useful when you want to switch branches, but you're in the middle of a change and aren't ready to commit.

16. How do you stash changes in Git?

Answer: You can stash changes using the command `git stash save "Your message"`. This command temporarily saves the changes, which can be re-applied later.

17. What is a .gitignore file?

Answer: A .gitignore file is a file that Git has been instructed to ignore. It should be placed in the root directory for the repository. This is useful for files or directories that you don't want Git to track or include in your repository's commits.

18. How do you revert a commit in Git?

Answer: You can revert a commit in Git using the command `git revert <commit-id>`. This command creates a new commit that undoes the changes made in the specified commit.

19. What is cherry-pick in Git?

Answer: Cherry-picking in Git means to choose a commit from one branch and apply it onto another. This is useful if you made a mistake and committed a change into the wrong branch, but don't want to merge the entire branch. You can do this with the command `git cherry-pick <commit-id>`.

20. What is the HEAD in Git?

Answer: HEAD is a reference to the last commit in the currently checked-out branch in Git. You can think of the HEAD as the "current branch".

21. How do you view the commit history in Git?

Answer: You can view the commit history in Git using the command `git log`. This command shows a list of all commits made to the current branch.

22. What is the difference between a "fork" and a "clone" in GitHub?

Answer: A "clone" is a copy of a repository that lives on your computer instead of on a website's server somewhere, or the act of making that copy. A "fork" is a copy of a repository that lives on your GitHub account instead of the account you're viewing, or the act of making that copy.

23. What is a bare repository in Git?

Answer: A bare repository in Git is a repository that only contains the .git folder and no working directory. It doesn't contain any checked-out source code files.

24. How do you squash commits in Git?

Answer: Squashing is the act of condensing multiple commits into a single commit. This is often done to clean up a messy history before merging a feature branch into main. You can squash commits using `git rebase -i HEAD~<number-of-commits>`.

25. What is Git bisect and how can it be useful?

Answer: Git bisect is a powerful tool in Git that uses a binary search algorithm to find which commit in your project's history introduced a bug. You use it by first telling it a "bad" commit that is known to contain the bug, and a "good" commit that is known to be before the bug was introduced. Git bisect then checks out a commit in the middle of that range, and you tell it whether the checked-out commit is good or bad. It continues narrowing down the range until it finds the exact commit that introduced the bug.

26. What are the different states of a file in Git?

Answer: In Git, a file can be in one of three states: modified, staged, and committed. Modified means that you have changed the file but have not committed it to your database yet. Staged means that you have marked a modified file in its current version to go into your next commit snapshot. Committed means that the data is safely stored in your local database.

27. How do you rename a branch in Git?

Answer: You can rename a branch in Git using the command `git branch -m <old-name> <new-name>`. If you are on the branch you want to rename, you can use `git branch -m <new-name>`.

28. What is the use of 'git config' command?

Answer: The 'git config' command is used to set or get configuration variables that control all aspects of how Git looks and operates. These variables can be stored in three different places: /etc/gitconfig file (contains values for every user on the system and all their repositories), ~/.gitconfig file (specific to your user), and the config file in the git directory (repo-specific config file).

29. How do you remove a file from Git without removing it from your file system?

Answer: You can remove a file from Git without removing it from your file system using the command `git rm --cached <file>`. This command stages the removal of the file from the repository, but leaves it on disk.

30. What is a conflict in Git and how can it be resolved?

Answer: A conflict in Git arises when two branches have changed the same part of the same file, and then those branches are merged together. For example, if you make a change on a branch, and someone else made a different change on a different branch to the same part of the same file, Git won't be able to figure out which change should take precedence. Conflicts are resolved by editing the files to fix the conflicting changes and then adding the resolved files by running `git add`, and then committing the resolved changes with `git commit`.

31. What is the difference between 'git stash pop' and 'git stash apply'?

Answer: When you execute 'git stash pop', it will apply the stashed changes to your current working directory and also remove the stashed item from your stash list. On the other hand, 'git stash apply' will apply the stashed changes and leave the stash in your stash list.

32. What is the use of 'git log --grep'?

Answer: The 'git log --grep' command is used to search for commits with a commit message that matches the pattern. This can be useful if you're looking for a specific commit but only remember part of the commit message.

33. What is the use of 'git reset'?

Answer: The 'git reset' command is used to reset your current HEAD to the specified state. It can be used to unstage files, undo commits, or to completely reset your repository to a previous commit.

34. What is the difference between 'git remote' and 'git clone'?

Answer: The 'git remote' command allows you to create, view, and delete connections to other repositories. 'git clone', on the other hand, is used to target an existing repository and create a clone, or copy of the target repository.

35. What is the use of 'git tag'?

Answer: 'git tag' is used to capture a point in history that is used to mark a specific point in the repository's history as being important. Typically, people use this functionality to mark release points (v1.0, and so on).

36. What is the use of 'git push' command?

Answer: The 'git push' command is used to push changes from the local repository to the remote repository. It transfers commits made on the local branch to a remote repository.

37. What is the use of 'git rm'?

Answer: The 'git rm' command is used to remove files from the staging area and the working directory. It tells Git to stop tracking the file and also removes the file from your working directory.

38. What is the use of 'git stash drop'?

Answer: The 'git stash drop' command is used to remove a stashed change. If you do not provide a stash, it will remove the latest one.

39. What is the use of 'git show'?

Answer: The 'git show' command is used to view information about any git object. By default, it shows the changes made in the last commit.

40. What is the use of 'git diff'?

Answer: The 'git diff' command is used to show changes between commits, commit and working tree, etc. It shows the changes which are not yet staged.

41. What is the use of 'git branch -d' and 'git branch -D'?

Answer: The 'git branch -d' command is used to delete a branch. It will not let you delete the branch if it has unmerged changes. On the other hand, 'git branch -D' is used to force delete the branch, regardless of whether it has unmerged changes or not.

42. What is the use of 'git fetch'?

Answer: The 'git fetch' command is used to download commits, files, and refs from a remote repository to your local repository. It fetches all the changes but does not merge them with your current branch.

43. What is the use of 'git merge'?

Answer: The 'git merge' command is used to merge branches. It takes the contents of a source branch and integrates it with the target branch.

44. What is the use of 'git rebase'?

Answer: The 'git rebase' command is used to move or combine a sequence of commits to a new base commit. It is a way to integrate changes from one branch into another.

45. What is the use of 'git cherry-pick'?

Answer: The 'git cherry-pick' command is used to apply the changes introduced by some existing commits. It applies the changes added by the commit at the tip of the current branch.

46. How would you revert a commit that has already been pushed and made public?

Answer: One way to revert a commit that has been pushed and made public is to use the `git revert <commit-id>` command. This will create a new commit that undoes the changes made in the specified commit. This is a safe way to undo changes, as it doesn't alter the existing commit history.

47. How would you resolve a bug in production while working on a new feature?

Answer: You can use Git branches to handle this situation. First, you would commit your changes on the feature branch. Then, switch to the production branch and create a new branch for bug fixing. After fixing the bug and testing it, you can merge the bug fix branch back to the production branch. Then, switch back to your feature branch to continue working on the new feature.

48. How would you handle a situation where you need to work on a project with a team of developers?

Answer: Git is designed to handle this scenario. Each developer would clone the repository and work locally on their own feature branches. When a developer is ready to share their work, they would push their branch to the remote repository and create a Pull Request. The team can then review the changes, discuss modifications if necessary, and then merge the changes into the main branch.

49. How would you keep your forked repository updated with the original repository?

Answer: You can do this by adding the original repository as a remote using `git remote add upstream <original-repo-url>`. Then, you can fetch all the branches from the upstream repository using `git fetch upstream`, and merge the changes from the upstream's main branch into your local main branch using `git merge upstream/main`.

50. How would you clean up your feature branch after it has been merged into the main branch?

Answer: After your feature branch has been merged and you no longer need it, you can delete it to keep your working directory clean. You can delete the local branch using `git branch -d <branch-name>` and the remote branch using `git push origin --delete <branch-name>`.

51. How would you handle a situation where you need to undo a commit that was made a while ago?

Answer: You can use the `git revert <commit-id>` command to undo an old commit. This will create a new commit that undoes the changes made in the specified commit. If you need to undo multiple commits, you can use the `git rebase -i` command and drop the commits you want to undo.

52. How would you handle a situation where you accidentally committed changes to the wrong branch?

Answer: You can use the `git cherry-pick` command to apply these changes to the correct branch. First, note down the commit id of the wrong commit. Then, switch to the correct branch and run `git cherry-pick <commit-id>`. This will apply the changes to the correct branch. Then, you can go back to the wrong branch and undo the wrong commit using `git reset --hard HEAD~1`.

53. How would you resolve a merge conflict in Git?

Answer: Merge conflicts occur when the same part of a file is modified in two different branches and then those branches are merged. To resolve the conflict, you need to manually edit the conflicted files to decide which changes to keep. After resolving the conflict, you add the resolved files to the staging area using `git add`, and then commit the resolved changes using `git commit`.

54. How would you handle a situation where you need to work on multiple features at the same time?

Answer: You can use Git branches to work on multiple features simultaneously. Create a separate branch for each feature. This way, you can switch between branches to work on different features. Once a feature is completed and tested, you can merge it back into the main branch.

55. How would you handle a situation where you need to temporarily switch from the task you are working on to another task?

Answer: You can use the `git stash` command to save your changes without committing them. This allows you to switch to another branch and work on a different task. When you are ready to go back to the original task, you can use `git stash pop` to reapply the stashed changes.

56. How would you handle a situation where you need to collaborate with a team on a project, but you don't want to share all the changes until they are complete?

Answer: You can use a fork of the main repository or a separate feature branch for this purpose. Make all the changes in your fork or feature branch. Once the changes are complete and tested, you can create a Pull Request to merge your changes into the main repository.

57. How would you handle a situation where you made a commit, but then realized you made a mistake in your commit message?

Answer: If you haven't pushed the commit to the remote repository yet, you can amend the commit message using `git commit --amend`. This allows you to modify the most recent commit message. If you have already pushed the commit, you would have to force push the amended commit using `git push --force`, but this can cause problems for others who have cloned or forked your repository, so use with caution.

58. How would you handle a situation where you need to find out when a specific change was made and who made the change?

Answer: You can use the `git blame <file>` command. This shows line-by-line annotation of the specified file, indicating which commit was the last to modify each line. This can help you find out when a specific change was made and who made the change.

https://www.interviewbit.com/git-interview-questions/