Hexaware Technologies

IMS DevOps

Interview Questions

Srikrishna prakash Sharma

Git Interview Questions

Level: Beginner

1. What is Git?

   Git is an Open-Source distributed version control system.

2. What are the advantages of Git?
   - Ease of use
   - High availability
   - Excellent support for distributed development
   - Community Support
   - Better and quicker release cycle
   - Superior disk utilization
   - Data redundancy and replication
   - Better team collaboration

3. What are the diverse ways to create repository using Git?
   - Create the repository using web console and clone it locally.
   - Initialize a local repository and set up to newly created remote repository.

4. What is commit?

   'Commit' updates local repository with references along with associated objects.

5. What is the difference between fetch and pull?

   The command `git fetch` retrieves updates from the remote repository to local repository without integrating them. However, the command `git pull` fetches the updates and also incorporates them into the active branch.

6. What are git reference logs or reflog? → **Intermediate**

   Git keeps track of updates to the tip of the branches using a mechanism called reference logs, or "reflogs."

7. What is a `. gitignore` file?

   This file instructs Git on the files or directories to exclude from the commit.

8. What is git merge?
   The command `git merge <source_branch>'will combine multiple sequences of commits into one unified history.

9. Difference between merge and rebase?
   The command `git merge' allows for the creation of a new "merge commit" occurring between the source and target commit. Whereas `git rebase' can reapply the source branch commits on top of the target branch.

10. What are the purposes of branching?
    The purpose of branching in GIT is to go to the previous work, keeping your recent work intact. At once able to merge the previous work and current work.

11. What is the command to delete a branch?
    The command Use `git branch -d <branch_name>` to delete a local branch. If the branch is not fully merged, you may need to use `-D` instead.

12. What is the command to create a branch locally?
    The command `git checkout -b <branch_name> ' creates branch locally.

13. What is Pull Request?
    It is feature of git services that allows a contributor to ask a maintainer of a repository to review and merge the code

14. What does `git reset` do? → **Intermediate**
The command `git reset` is used to undo changes. It has three main modes: `--soft`, `--mixed`, and `--hard`.

15. How do you amend a commit message? → **Intermediate**
Use `git commit --amend` to change your most recent commit message.

16. How do you list all the remote repositories configured?
Use `git branch -r` to list all remote branches.

17. How do you find a commit by a message? → **Intermediate**
Use `git log --grep=<search-pattern>` to search through commit messages

18. How do you track changes between two commits?
Use the command ` git diff HEAD'

19. What is a fast-forward merge in Git?
A fast forward in Git occurs during a merge operation when the base branch that's being merged into has no new commits since the feature branch (the branch being merged) was created or last updated. Instead of creating a new "merge commit," Git simply moves the pointer forward, hence the term "fast forward."

20. What is a detached HEAD in Git? → **Intermediate**
A detached HEAD occurs when you check out a commit, branch, or tag that is not the latest commit of a branch.

21. How do you squash the last N commits into a single commit? → **Intermediate**
Use `git rebase --interactive HEAD~N` and choose `squash` for the commits you want to combine.

22. What are submodules in Git? → **Advanced**

Submodules enable the inclusion of one Git repository within another as a subdirectory. This feature is beneficial for integrating external projects or libraries into your main project.

23. What is `git bisect`? → **Advanced**

The command `git bisect` assists in identifying the commit responsible for introducing a bug through the application of a binary search algorithm.

24. How do you view the commit history?

The command 'git log' to view the commit history.

25. What is `git merge --squash`? → **Intermediate**

The `git merge --squash` command consolidates all commits from a specified feature branch into a single commit when merging into the target branch

26. What is the significance of `git push --force`? → **Intermediate**

The command `git push --force` is used to overwrite the remote history with your local history. It should be used with caution as it can overwrite changes in the remote repository.

27. What is a bare repository in Git? → **Advanced**

A bare repository is created without a working tree; you must mention the "--bare" flag explicitly to create one. You can use the bare repository as the remote repository shared among several people. You cannot make any commits in a bare repository, or track changes made in projects by a bare repository. Central repositories use bare repositories because Git does not allow you to push.

A bare repository is a Git repository that does not have a working directory, making it suitable for sharing code as it contains only the version history.

28. What is the purpose of `git tag -a`? → **Intermediate**

The command `git tag -a` creates an annotated tag, which includes metadata such as the tagger name, email, and date, useful for marking releases.

29. How do you find a list of files that have changed in a specific commit? → **Advanced**

The command `git show --name-only <commit-hash>` to list the files that changed in a commit. The current commit hash can be retrieved by running the command `git rev-parse HEAD or git log --pretty=format:'%h' -n 1 ‘

30. What is the function of ‘git rm’?

To remove the file from the staging area and also off your disk ‘git rm’ is used.

31. What is the difference between `HEAD`, `working tree` and `index` in Git? → **Advanced**

HEAD` refers to the last commit on the current branch, `working tree` is the set of files in your directory, and `index` (or staging area) is a staging area for commits.

32. What does `git fetch --prune` do? → **Advanced**

The command `git fetch --prune` removes remote-tracking branches that no longer exist on the remote.

33. How do you list all the remote branches?

Use the command `git branch -a’

34. How do you copy a commit from one branch to another? → **Intermediate**

Use the command `git cherry-pick <commit-hash>` to apply the changes from a commit on another branch to the current branch.

35. How do you compare two branches in Git?

Use `git diff <branch1>..<branch2>` to see the differences between the two

36. What is `git reset --soft`?

The command `git reset --soft <commit-hash>` undoes commits but keeps the changes in the staging area.

37. What is git ls-tree? → **Intermediate**

The command 'git ls-tree' represents a tree object including the mode and the name of each item and the SHA-1 value of the blob or the tree.

38. How can you fix a broken commit? → **Advanced**

To fix any broken commit, you will use the command "git commit—amend". By running this command, you can fix the broken commit message in the editor.

39. What is the difference between Git and GitHub?

Git is a command-line tool that's installed locally on a system. GitHub is a graphical user interface that's hosted on the web

40. What command do you use to return to a previous commit? → **Advanced**

To return to a previous commit on a private, local repository, you'll first want to run git log to pull up the branch's history. Then, after identifying the version's hash you want to go to, use the git reset command to change the repository to the commit.

For a public, remote repository, the git revert command is a safer option. It'll create a new commit with the previous edits rather than delete commits from the history.

41. What methods do you use to clean up your feature branches? → **Intermediate**

git merge --squash is a command that can merge multiple commits of a branch.

git commit --fixup marks the commit as a fix of the previous commit.

git rebase -i --autosquash is a rebase type of squash for merging multiple commits

42. Explain Git hooks.

Git hooks are custom scripts that can run when certain actions occur. There are two types: client-side hooks and server-side hooks. Git hooks can be written in any scripting programming language. They should be located within the hooks subdirectory of the. git directory.

43. I made changes in three branches. What is the command to push all three branches?

The command is `git push –all'

44. How do you include tags in the push command?

The command `git push <remote> --tags'

45. What is the purpose of `git worktree`? → **Intermediate**

The command `git worktree` allows you to have multiple working trees attached to the same repository, enabling you to work on multiple branches simultaneously without switching the current worktree.

46. What is the significance of `git push --force-with-lease` over `git push --force`? → **Intermediate**

The command `git push --force-with-lease` ensures that you do not overwrite any work on the remote repository that you haven't seen, unlike `git push --force` which overwrites the remote changes blindly.

47. How do you change the URL of a remote repository? → **Advanced**

    Use `git remote set-url <remote-name> <new-url>` to change the URL

48. How do you clean untracked files from your working directory? → **Intermediate**

    Use `git clean` to remove untracked files from your working directory.

49. How do you see the difference between staged files buy not committed? → **Intermediate**

    Use the git diff –staged

50. How do you revert a commit that has already been pushed and made public?

    Use the command `git revert <commit id>'

51. Name the few branching strategies? → **Intermediate**

    GitFlow, GitHubFlow and Trunk-based development.

52. How do you recover a deleted branch? → **Advanced**

    Execute the command git reflog and git checkout -b <branch-name> <sha1-of-commit>

53. How do you find a commit that broke something after a merge operation? → **Advanced**

    Use the git-bisect command based on the binary search and run the following commands:

git bisect start #: initiates bisecting session

git bisect bad #: marks current revision as bad

git bisect good revision #: marks last known commit as good revision

Once you run the above commands, git will check out a revision labeled as halfway between "good" and "bad" versions. You can run this step again by marking the commit as "good" or "bad" and the process continues until the commit with a bug is found

54. How do you squash the last N commits into a single commit? → **Advanced**

Squash refers to combining two or more commits into one. Use the following command to write a new commit message from the beginning:

git reset -soft HEAD~N &&git commit

55. How do you reapply a commit that has been reverted? → **Advanced**

Use `git revert <commit-hash>` to revert the revert commit, effectively reapplying the original commit.

56. How do you integrate changes from a remote branch without a merge commit?

Use `git pull --rebase` to rebase the current branch on top of the remote branch, integrating changes without a merge commit

57. What is a symbolic reference in Git?

A symbolic reference is a reference to another reference, such as a branch name being a reference to a commit. The HEAD is a common example of a symbolic reference.

58. What is the difference between a shallow clone and a deep clone in Git? → **Advanced**

A shallow clone (`git clone --depth 1`) creates a copy of the repository with a limited history depth, reducing time and space, whereas a deep clone includes the complete history.

59. Explain the difference between `git stash pop` and `git stash apply`. → **Advanced**

The command `git stash pop` implements the modifications from the most recent stash and then deletes it from the stash inventory, while `git stash apply` enacts the changes yet preserves them within the stash inventory.

60. What is the purpose of `git fsck` and how do you use it? → **Advanced**

The command `git fsck` (file system check) is used to verify the integrity of the Git file system, checking for corrupt objects.

61. How do you create a patch with Git? → **Advanced**

Use `git diff > patch_name.patch` to create a patch file with the changes between commits or branches.

62. What is the use of `git revert --no-commit`? → **Advanced**

`git revert --no-commit` reverts the changes made by one or more commits without committing the revert, allowing you to make additional changes before committing.

63. What is the significance of the `git fetch --tags` command? → **Intermediate**

The command `git fetch --tags` fetches all tags from the remote repository, updating your local tag references.

64. What is the Git object model? → **Advanced**

The Git object model consists of blobs (representing file data), trees (which organize the file structure), commits (which record the changes), and tags (which mark specific points in history).

65. What is the role of the `.git/index` file? → **Intermediate**

The `.git/index` file acts as the staging area (or index) for Git. It tracks which changes are staged for the next commit.

66. What are the advantages of using a rebase over a merge? → **Intermediate**

Rebase creates a cleaner, linear commit history, which can simplify the understanding and exploration of project history. It avoids unnecessary merge commits and can make the history easier to navigate.

67. How do you find and restore a deleted file in the Git history? → **Advanced**

Use `git log -- <file-path>` to find the commit where the file was deleted, then use `git checkout <commit-hash>^ -- <file-path>` to restore it.

68. What are the advantages of using a distributed version control system?

Fast: All work can be done by different developers on the local repository, requiring no call to the server.

Easy branching and merging: Since the codebase resides on the local hard disk, branching and merging will be easier than centralized VCS.

Local branching: Developers can create, work on, and merge as many local branches as they want. Once the merging is done, developers can delete the local branches, so they will not be visible to anyone.

Snapshots instead of differences: You can have the complete code repository for each performed commit using snapshots. It

allows you to revert back any commit without making the changes to the base version.

Scalable: A DVC system is highly scalable compared to a centralized VCS, especially for open-source projects allowing millions of developers to contribute.

69. What is head in terms of Git? → **Advanced**

The head is a reference pointing to the tip (latest commit) of a branch. To check your repository's head, go to the path .git/refs/heads/. The path will offer one file per branch, displaying the commit ID of that branch's tip (most recent commit). While the HEAD is a special ref pointing to the commit you're currently working on. To put it simply, head is nothing but a global variable or environmental variable in your repository. Based on the commit, you made to your working directory and checked out, the value of head changes. A file called. git/HEAD stores this head.

Concepts

Fast forward merge vs. regular merge

Fast-forward merges

Fast-forward merges occur when there is a linear path from the base branch to the target branch, allowing the head of the base branch to be simply moved forward to point to the target branch. The target branch's pointer is moved forward to the head of the source branch if no divergent work has been done on the target branch. This effectively brings the history of the target branch in line with the source branch.

Pros:

Simpler history: Fast-forward merges keep the repository history linear and clean, which can simplify the understanding and navigation of the history.

Less overhead: No extra merge commits are created, which can be preferable in small projects or projects with fewer parallel developments.

Cons:

Loss of context: The major downside is the loss of context regarding the feature's development process. Without a dedicated merge commit, it's harder to see the boundary between different features or to identify when a feature was merged.

Complicated reverts: Reverting a specific feature becomes more difficult as there's no single merge commit to revert. You would need to individually revert all commits pertaining to that feature.

Non-fast-forward merges (--no-ff)

Using the --no-ff flag forces a merge commit even when a fast-forward merge would be possible. This approach is preferred in the git-flow mode introduced by Vincent Driessen, for merging feature, release, and hotfix branches back to develop or main branches.

Pros:

Preserves history: It keeps a clear record of project changes, including the context of feature development and integration.

Easier to manage features: Features can be collaboratively developed and then cleanly merged as a unit, which also simplifies potential reverts to a single merge commit.

Facilitates code review and collaboration: The merge commits provide a natural place for code review and discussions about specific features or fixes.

Cons:

Complex history: This method introduces additional merge commits that can clutter the project history, potentially making it harder to follow.

More merge conflicts: There is a slightly increased likelihood of encountering merge conflicts that need to be resolved manually, as changes from the base branch are not automatically incorporated into the feature branch during its development.