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

**Answer:** Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It allows multiple developers to work on the same project without overwriting each other's changes, maintains a history of changes, and facilitates collaboration.

**2. What is a repository in Git?**

**Answer:** A repository (or repo) in Git is a storage location for your project's code and history of changes. It can be local (on your computer) or remote (on a server). The repository contains all your project's files, as well as information about the history of changes to those files.

**3. Explain the difference between git pull and git fetch.**

**Answer:** git fetch downloads commits, files, and refs from a remote repository into your local repository, updating your remote-tracking branches. However, it doesn't merge any changes into your working directory. git pull is essentially git fetch followed by git merge. It fetches the changes and then merges them into your current branch.

**4. How do you create a new repository in Git?**

**Answer:**

* To create a new repository locally, you use:

sh

Copy code

git init

* To clone an existing remote repository, you use:

sh

Copy code

git clone <repository\_url>

**5. What is the difference between git clone and git init?**

**Answer:** git init initializes a new, empty repository in the current directory. It’s used to start a new project repository. git clone creates a copy of an existing repository, including all files, branches, and history.

**Intermediate Questions**

**6. Explain branching in Git. Why is it useful?**

**Answer:** Branching in Git allows you to diverge from the main line of development and continue to work without affecting that main line. Branches are useful for developing features, fixing bugs, or experimenting with new ideas. They provide an isolated environment for your changes until you're ready to merge them back into the main branch.

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

**Answer:**

* To create a new branch:

sh

Copy code

git branch <branch\_name>

* To switch to that branch:

sh

Copy code

git checkout <branch\_name>

* Or you can do both in one command:

sh

Copy code

git checkout -b <branch\_name>

**8. What is a merge conflict, and how do you resolve it?**

**Answer:** A merge conflict occurs when Git cannot automatically resolve differences in code between branches. This happens when the same part of a file has been changed in different ways on two branches. To resolve it, you need to manually edit the conflicting files to choose which changes to keep, then mark the conflict as resolved with:

sh

Copy code

git add <resolved\_file>

After resolving all conflicts, complete the merge with:

sh

Copy code

git commit

**9. Explain the use of git rebase. How is it different from git merge?**

**Answer:** git rebase re-applies commits from your current branch onto another base branch. It creates a linear history by moving the entire branch to begin on the tip of another branch. git merge creates a new commit that brings together the changes from two branches, resulting in a non-linear history. Rebase is often used to keep a clean project history, while merge is used to combine completed work from different branches.

**10. What is a pull request, and how do you create one?**

**Answer:** A pull request is a way to propose changes to a repository. It allows developers to notify others about changes they've pushed to a branch in a repository. To create a pull request:

* Push your branch to the remote repository:

sh

Copy code

git push origin <branch\_name>

* Go to the repository on GitHub/GitLab/Bitbucket and navigate to the "Pull Requests" tab.
* Click "New Pull Request", select your branch, and submit your pull request with a description of the changes.

**Advanced Questions**

**11. What are some best practices for writing commit messages?**

**Answer:**

* Keep the first line of the commit message under 50 characters.
* Use the imperative mood (e.g., "Fix bug" rather than "Fixed bug" or "Fixes bug").
* Provide a detailed description after the first line if necessary.
* Explain the why behind the change, not just the what.

**12. Explain the concept of Git hooks. Have you used any? If so, which ones and for what purposes?**

**Answer:** Git hooks are scripts that run automatically on certain Git events like commits, merges, and pushes. They allow you to customize Git’s behavior. Examples include:

* pre-commit: to check code style or run tests before committing.
* pre-push: to run tests or checks before pushing.
* post-merge: to recompile code or notify team members after a merge.

**13. How do you handle large binary files in a Git repository?**

**Answer:** Large binary files can be managed using Git LFS (Large File Storage). It replaces large files with text pointers inside Git, while storing the file contents on a remote server. To use Git LFS:

sh

Copy code

git lfs install

git lfs track "\*.psd"

git add .gitattributes

git add <large\_file>

git commit -m "Add large file with LFS"

git push

**14. What strategies do you use to manage dependencies in a Git repository?**

**Answer:**

* Use a dependency manager like npm, yarn, pip, or Maven.
* Store dependency version files (e.g., package.json, requirements.txt) in the repository.
* Avoid committing dependencies directly to the repository. Instead, use .gitignore to exclude them and allow the dependency manager to handle them.

**15. Describe the git stash command and provide a use case for it.**

**Answer:** git stash temporarily shelves (or stashes) changes you’ve made to your working directory so you can work on something else and come back to them later. Use case: You're working on a feature and suddenly need to switch to another branch to fix a bug. You can stash your changes:

sh

Copy code

git stash

git checkout <other\_branch>

When you return to your feature branch, reapply your changes with:

sh

Copy code

git stash apply

**Scenario-Based Questions**

**16. You need to revert a commit that is in the middle of a branch. How would you approach this?**

**Answer:** Use git revert to create a new commit that undoes the changes from the specific commit:

sh

Copy code

git revert <commit\_hash>

This method safely reverses the changes without rewriting history.

**17. How do you handle a situation where your local branch is behind the remote branch and you have local changes?**

**Answer:**

* First, stash your local changes:

sh

Copy code

git stash

* Pull the latest changes from the remote branch:

sh

Copy code

git pull

* Apply your stashed changes:

sh

Copy code

git stash apply

* Resolve any conflicts, if necessary, and commit your changes.

**18. Describe a time when you encountered a difficult merge conflict. How did you resolve it?**

**Answer:** Describe a specific situation where you faced a merge conflict. Explain the steps you took to understand the conflicting changes, how you communicated with the team to resolve discrepancies, and the tools or commands you used (like git mergetool, git diff) to resolve the conflict and successfully complete the merge.

**19. How would you set up a Git workflow for a large team with multiple features being developed simultaneously?**

**Answer:**

* Use a branching strategy like Gitflow or feature branching.
* Ensure there is a develop branch for ongoing integration and a master branch for stable releases.
* Each feature should be developed in its own branch.
* Use pull requests for code reviews and discussions before merging into develop.
* Regularly integrate and test changes in the develop branch to catch issues early.

**20. You need to find a specific bug introduced in the last 100 commits. How would you go about it?**

**Answer:** Use git bisect to perform a binary search through the commit history:

sh

Copy code

git bisect start

git bisect bad HEAD

git bisect good <commit\_hash\_of\_known\_good\_state>

Git will checkout commits between the known good and bad commits. You test each commit, mark it as good or bad with git bisect good or git bisect bad, and Git continues the process until it identifies the commit that introduced the bug.

**Practical Questions**

**21. How do you squash multiple commits into a single commit?**

**Answer:**

* Use interactive rebase:

sh

Copy code

git rebase -i HEAD~<number\_of\_commits>

* Mark the commits you want to squash with s (for squash), then save and close the editor. This combines the commits into a single one.

**22. How do you cherry-pick a commit from one branch to another?**

**Answer:**

* First, checkout the branch where you want to apply the commit:

sh

Copy code

git