1. What is Git and why is it used?

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 a project simultaneously without overwriting each other's changes. Git tracks changes to files, allowing developers to revert to previous versions and collaborate efficiently by merging contributions from different sources.

2. Explain the difference between Git pull and Git fetch.

- **Git fetch** downloads commits, files, and references from a remote repository into your local repository without integrating them into your working directory. It updates your local copy of the repository's metadata.
- **Git pull** does both fetching and merging. It downloads the latest changes from the remote repository and immediately tries to merge them into your current branch.

3. How do you revert a commit in Git?

To revert a commit in Git, you can use the `git revert` command, which creates a new commit that undoes the changes of a previous commit:

- `git revert <commit_hash>`

Alternatively, to remove the commit from history (which is more destructive), you can use `git reset`:

- `git reset --hard <commit_hash>`

Be cautious with `git reset --hard` as it can rewrite history and discard changes.

4. Describe the Git staging area.

The Git staging area, or index, is an intermediate area where commits can be formatted and reviewed before completing the commit. It's a place to collect changes that will go into the next commit. You add files to the staging area using `git add <file>`.

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

A merge conflict occurs when Git is unable to automatically reconcile differences in code between two commits. This often happens when changes are made to the same line of a file or when one branch deletes a file that the other branch modifies. To resolve a merge conflict, you must manually edit the conflicting files to choose which changes to keep, and then mark the conflict as resolved with:

- git add <resolved_file>
- git commit

6. How does Git branching contribute to collaboration?

Git branching allows multiple developers to work on different features or bug fixes simultaneously without interfering with each other's work. Each branch can be developed, tested, and integrated independently, making it easier to manage and review changes. Branches can later be merged into the main codebase, facilitating collaboration and parallel development.

7. What is the purpose of Git rebase?

Git rebase is used to move or combine a sequence of commits to a new base commit. It helps in maintaining a clean project history by avoiding unnecessary merge commits. Rebasing re-applies commits from a branch onto another base tip, allowing for linear history and easier code review.

8. Explain the difference between Git clone and Git fork.

- **Git clone** creates a local copy of a remote repository. It is typically used by a developer to get a copy of the repository to work on.
- **Git fork** is a server-side copy of a repository. It creates a separate copy of the original repository on the user's Git hosting account (like GitHub) to make changes independently. Forks are often used to contribute to someone else's project.

9. How do you delete a branch in Git?

To delete a local branch, use:

- git branch -d <branch_name>
- git branch -D <branch_name>

- git push origin --delete <branch_name>

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

Git hooks are custom scripts that are triggered by certain Git events like commits, merges, and pushes. They allow automation of tasks, such as code quality checks, automated testing, and enforcing commit message formats. Hooks are stored in the `.git/hooks` directory and can be enabled by creating executable scripts with the appropriate names (e.g., `pre-commit`, `post-commit`).