1. What is Git and how does it differ from GitHub?

Ans: Git is a distributed version control system used for tracking changes in source code during software development, while GitHub is a web-based hosting service for Git repositories, providing additional collaboration and project management features.

1. Explain the purpose of branches in Git and GitHub. How do you create and merge branches?

Ans: Branches in Git and GitHub are parallel versions of the codebase, allowing developers to work on features or fixes independently. You create a new branch using the **git branch** command and merge branches using **git merge**.

1. What is a pull request? How does it facilitate collaboration in GitHub?

Ans: A pull request in GitHub is a method of submitting contributions to a repository for review and merging. It facilitates collaboration by allowing developers to discuss changes, review code, and propose improvements before merging into the main codebase.

1. Describe the difference between Git fetch and Git pull. When would you use each command?

Ans: **Git fetch** retrieves changes from a remote repository but does not merge them into the current branch, while **git pull** fetches changes and merges them into the current branch. You'd use **git fetch** to inspect changes before merging, and **git pull** to fetch and merge changes in one step.

1. How do you revert a commit in Git? Explain with an example.

Ans: You can revert a commit in Git using the **git revert** command followed by the commit hash. This creates a new commit that undoes the changes introduced by the specified commit.

1. Describe the Git workflow you would use for collaborating on a project with a team using GitHub. Include branching, merging, and pull requests in your explanation.

Ans: A typical Git workflow for collaborating on a project involves creating feature branches for new features or bug fixes, pushing branches to a shared repository, reviewing changes via pull requests, merging branches into the main branch, and resolving conflicts as needed.

1. What are Git hooks in GitHub? How can you use pre-commit and post-commit hooks to automate tasks in your Git workflow?

Ans: Git hooks are scripts that run automatically at specific points in the Git workflow. Pre-commit hooks run before a commit is made, while post-commit hooks run after a commit is made. You can use them to automate tasks like linting, testing, or code formatting.

1. Explain the difference between Git rebase and Git merge. When would you use each command?

Ans: **Git rebase** rewrites the commit history by moving, combining, or deleting commits, while **git merge** combines changes from different branches into the current branch. You'd use **git rebase** for a cleaner commit history and **git merge** for preserving the history of changes.

1. Discuss the purpose of `.gitignore` file in a Git repository. How can you use it to exclude certain files or directories from version control?

Ans: The **.gitignore** file specifies intentionally untracked files that Git should ignore. You can use it to exclude certain files, directories, or patterns from version control, such as build artifacts, temporary files, or sensitive information.

1. How can you revert a commit that has already been pushed to a remote repository in GitHub? Explain the steps involved.

Ans: To revert a commit that has already been pushed to a remote repository, you'd first revert the commit locally using **git revert**, then push the reverted commit to the remote repository using **git push**.