Top 100+ Git Interview Questions [2024]: Crack the Code

Lesson 11 of 12

By Simplilearn

Last updated on Feb 13, 2024

423983

Previous

Next

Tutorial Playlist

Table of Contents

Git stands as the leading tool for managing source code. It serves both programmers and individuals without technical expertise, offering collaboration tools such as issue tracking, project management, and wikis. In DevOps practices, Git's role is pivotal. Given Git's prominence in the DevOps sphere, a surge in job prospects is anticipated. A Grand View Research study forecasts the worldwide DevOps market to hit $12.85 billion by 2025, expanding at an 18.60% annual growth rate.

What Is Git and Why Is It Used?

Git is a widely adopted distributed version control system (VCS) essential for managing modifications in source code throughout the software development process. Linus Torvalds developed it in 2005 to facilitate the Linux kernel's development. Engineered for speed and efficiency, Git is versatile enough to manage projects of any size, from modest to expansive.

Key Features of Git

1. Distributed Architecture: In contrast to centralized version control systems, Git employs a distributed framework. This means that each developer's local copy of the codebase doubles as a repository capable of storing the complete history of modifications. This approach not only offers redundancy but also enhances performance.
2. Data Integrity: Git ensures the integrity of version control data by combining cryptographic hash functions (SHA-1), ensuring the history of project files and commits is secure and unchanged over time.
3. Branching and Merging: Git provides robust tools for branching and merging, allowing developers to work on different features or fixes in parallel before incorporating them into a main line of development. This promotes a more fluid workflow and helps to manage changes effectively.
4. Speed: Git is designed to handle operations quickly, regardless of the project size. Many operations are performed locally, reducing the need for network access and speeding up the workflow.
5. Open Source: Git is open source, meaning it's free to use, and its source code is available for anyone to inspect, modify, and enhance.

Why Is Git Used?

1. Version Control: Git enables several developers to collaborate on a single project at the same time without hindering one another's contributions. It meticulously records every modification to the project's codebase, identifying the author and the timestamp of each change. This functionality is vital for synchronizing team activities and addressing any arising disputes.
2. Backup and Restore: Changes in Git are committed to the local repository, creating a complete project history. This history provides a backup of every change, and it's possible to revert to any previous project state.
3. Collaboration: Git's distributed nature makes it easier for teams to collaborate on projects. With platforms like GitHub, GitLab, and Bitbucket, teams can share their repositories, review code, and manage projects together.
4. Branching and Experimentation: Developers can use branches in Git to experiment with new ideas or work on new features without affecting the main project. This encourages innovation and allows for testing new ideas in a controlled environment.
5. Track Changes: Git allows users to see precisely what changes have been made, by whom, and when. This is valuable for understanding a project's evolution and auditing changes.
6. Continuous Integration and Delivery (CI/CD): Git integrates seamlessly with numerous Continuous Integration/Continuous Deployment (CI/CD) solutions, streamlining software testing and deployment. This enhances the efficiency of the development workflow and minimizes the likelihood of mistakes.

Preparation Tips to Crack the Git Interview

1. Understand Core Concepts

Learn the Basics: Ensure you understand fundamental concepts like version control, the difference between Git and other version control systems (VCS), and their importance in software development.

Master Git Commands: Be comfortable with common Git commands (`git init`, `git clone`, `git add`, `git commit`, `git push`, `git pull`, `git branch`, `git checkout`, `git merge`, `git rebase`, etc.). Understand what each command does and when to use it.

Know Git Workflows: Familiarize yourself with different Git workflows, such as Git Flow, GitHub Flow, and GitLab Flow. Understand the pros and cons of each and scenarios where they are best applied.

2. Hands-on Practice

Use Git Regularly: Practice using Git for your personal or professional projects. Regular use will help reinforce your knowledge and make you comfortable with various commands and workflows.

Experiment with Branching and Merging: Create branches, merge them, and resolve conflicts. Understanding how to manage branches and merge them without conflicts is crucial.

Contribute to Open Source Projects: Engage with open source projects on platforms like GitHub. This gives you practical experience with collaboration tools and workflows used in the industry.

3. Dive into Advanced Topics

Understand Internals: Learn about the internals of Git, such as the .git directory, the staging area, and how Git stores changes. Knowledge of these areas can set you apart from other candidates.

Learn Advanced Commands: Get comfortable with advanced Git commands like `git stash`, `git rebase --interactive`, `git bisect`, `git cherry-pick`, and using hooks.

Explore Git Tools and Extensions: Familiarize yourself with tools and extensions that enhance Git’s capabilities, such as Git LFS (Large File Storage), GUI clients, and IDE integrations.

4. Review and Understand Common Errors

Familiarize with Common Issues: Understand common Git issues and errors, such as merge conflicts, detached HEAD state, and how to revert changes or recover lost commits.

Practice Troubleshooting: Diagnosing and fixing issues is as important as knowing how to avoid them. Practice troubleshooting common scenarios.

5. Prepare for Behavioral Questions

Collaboration Scenarios: Be ready to discuss how you have used Git in team settings, handled merge conflicts, and used Git to improve collaboration or workflow in your projects.

Contribution Examples: Be prepared to talk about specific contributions you've made to projects using Git, highlighting your understanding of version control best practices.

6. Stay Updated

Follow Git and Development Blogs: Follow relevant blogs and spaces to stay informed about the latest Git features and best practices.

Explore Git Documentation: The official Git documentation is an excellent resource for deepening your understanding and staying updated on new features.

7. Mock Interviews

Practice with Peers or Mentors: Conduct mock interviews to become comfortable articulating your thoughts and knowledge about Git.

Use Online Platforms: Platforms like Pramp or LeetCode offer mock interviews and problems to solve, which can help you practice in a simulated interview environment.

8. Review Interview Questions

Study Common Questions: Review common Git interview questions and prepare your answers. This includes both theoretical questions and practical exercises.

9. Showcase Your Knowledge

Create a Portfolio: If possible, have a portfolio of projects or contributions to showcase your practical experience with Git. This can be a GitHub profile or a personal website.

Basic Git Interview Questions and Answers

1. What is Git?

Git is a decentralized version control system engineered for rapid and effective management of projects ranging from modest to vast in scale. It supports collaborative efforts among developers by monitoring file modifications and enhancing teamwork.

2. What is a repository in Git?

A Git repository is a location where your project resides, acting as a storage area. This repository can exist locally within a directory on your computer or on a cloud-based platform like GitHub. It encompasses all the files associated with the project, along with a record of the modifications made to these files over time.

3. What is the difference between Git and GitHub?

Git is a tool for version control that enables you to monitor and record the evolution of your source code. GitHub is an online hosting service that facilitates the management of Git repositories. GitHub offers a user-friendly web interface along with functionalities such as permission management, task organization, error tracking, and the capability to handle feature suggestions.

4. How does Git work?

Git works by taking snapshots of a project's files. Unlike other version control systems, Git records the entire contents of each file and its changes every time a commit is made. This makes operations like branching, merging, and reverting changes more efficient.

5. What is a commit in Git?

In Git, a commit is a process that records a version of the project's presently prepared modifications. This record includes details on the modifications implemented, a distinctive identifier (a SHA-1 hash), the creator's identity, and the timestamp of the commit.

6. What is branching in Git?

In Git, branching allows you to veer off from the primary development path and proceed with separate tasks without impacting the main workflow. This technique enables the isolated development of features, bug fixes, or experimentation within a specific section of the repository, ensuring that each process remains distinct from the others.

7. What is a merge in Git?

Merging is a Git operation that integrates changes from one branch into another. It can be a fast-forward merge, where the target branch is updated to the latest commit of the source branch, or a three-way merge, where divergent branch histories are combined into a new commit.

8. What is a conflict in Git?

A conflict in Git occurs when two branches have made edits to the same line in a file or when one branch deletes a file while the other branch modifies it. Git cannot automatically resolve these changes; the developer must manually resolve the conflicts.

9. What is a pull request?

A pull request serves as a mechanism for contributing to a project, typically utilized in projects hosted on GitHub. In this process, a developer implements modifications within their own branch, uploads these changes to a repository, and then initiates a pull request. This action prompts the project's maintainers to examine the proposed changes, engage in discussions about possible adjustments, and ultimately integrate the pull request into the primary branch.

10. What is `git fetch` vs. `git pull`?

`git fetch` downloads updates from a remote repository to your local repository without integrating them. On the other hand, `git pull` not only fetches the updates but also incorporates them into your active branch.

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

To undo the modifications introduced by a previous commit while ensuring safety for public commits, employ `git revert <commit_hash>`, which generates a fresh commit that reverses the earlier changes. On the other hand, `git reset` allows reverting to an earlier state; however, exercise caution when applying it to public commits, as it alters the commit history.

12. What is a `.gitignore` file?

A `.gitignore` file is a textual document instructing Git on the files or directories to exclude from a project. This commonly encompasses files created during the build phase, local setup files, or files with confidential data.

13. How do you clone a repository?

To create a local copy of a repository on your machine, execute the command `git clone <repository_url>`. This action duplicates the repository locally.

14. What is `git stash`?

The `git stash` command temporarily stores your working directory's modifications, allowing you to switch branches without discarding your current progress.

15. How do you view the commit history?

Use `git log` to view the commit history. There are many options to customize the output, such as `git log --oneline` for a condensed view.

16. What is a remote in Git?

A Git remote refers to a shared repository utilized by all team members for the purpose of exchanging their updates. Typically, this repository is on a server or a cloud-based hosting service like GitHub.

17. How do you create a new branch?

Use `git branch <branch_name>` to create a new branch. Use `git checkout -b <branch_name>` to create and switch to it in one step.

18. What is `git merge --squash`?

The `git merge --squash` command consolidates all commits from a specified feature branch into a single commit when merging into the target branch, resulting in a tidier project history.

19. How do you resolve a merge conflict?

To resolve a merge conflict, edit the files to fix the conflicting changes. Then, use `git add` to stage the resolved files and `git commit` to commit the resolved merge.

20. What is `git rebase`?

`git rebase` transfers modifications from one branch to another, enabling the creation of a streamlined project timeline by relocating the updates of a feature branch to the forefront of the main branch.

21. What is the difference between `git merge` and `git rebase`?

The main difference is in how the branch history is presented. `git merge` preserves the history of a feature branch by creating a new merge commit. `git rebase` rewrites the feature branch's history to appear as if it was developed from the latest main branch, creating a linear history.

22. How do you change the last commit?

Use `git commit --amend` to modify the most recent commit. This can change the commit's message or include new changes.

23. What is `git push`?

`git push` is used to upload local repository content to a remote repository. It transfers commits from your local repo to the remote.

24. How do you delete a branch?

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. To delete a remote branch, use `git push <remote_name> --delete <branch_name>`.

25. What is `git checkout`?

`git checkout` allows navigating between different branches or reverting working tree files to a previous state. However, in the latest versions of Git, it is advised to use `git switch` for changing branches and `git restore` to revert files, each designated for these specific functions.

26. How do you list all the remote repositories configured?

Use `git remote -v` to list all the remote repositories configured for your local repository.

27. How do you add a file to the staging area?

Use `git add <file_name>` to add a file to the staging area, making it ready for a commit.

28. How do you remove a file from Git but not delete it from your file system?

Use `git rm --cached <file_name>` to remove a file from Git without deleting it from your filesystem.

29. What is `git diff`?

`git diff` shows the differences between files in the working directory and the index, or between commits.

30. How do you rename a Git branch?

To rename the current branch, use `git branch -m <new_name>`. To rename a different branch, use `git branch -m <old_name> <new_name>`.

31. What does `git reset` do?

`git reset` is used to undo changes. It has three main modes: `--soft`, `--mixed`, and `--hard`.

32. How do you amend a commit message?

Use `git commit --amend` to change your most recent commit message.

33. What is the HEAD in Git?

HEAD is a reference to the last commit in the currently checked-out branch.

34. How do you find a commit by a message?

Use `git log --grep=<search-pattern>` to search through commit messages.

35. What is the difference between `git stash pop` and `git stash apply`?

`git stash pop` applies stashed changes and removes them from the stash. `git stash apply` applies stashed changes but keeps them in the stash.

36. How do you list all branches that contain a specific commit?

Use `git branch --contains <commit>`.

37. What is a fast-forward merge in Git?

A fast-forward merge happens when the target branch's tip is behind the merged branch's tip, allowing the target branch to "catch up" by just moving forward to the merged branch's tip.

38. How do you create an empty commit?

Use `git commit --allow-empty` to create a commit with no changes.

39. How do you switch branches in Git?

Use `git checkout <branch-name>` to switch branches.

40. How do you ignore changes in a tracked file?

Use `git update-index --assume-unchanged <file>` to ignore changes in a tracked file temporarily.

Intermediate Git Interview Questions and Answers

1. Explain the Git branching strategy you use.

A common strategy is the Git Flow, which involves having a master branch, develop branch, feature branches, release branches, and hotfix branches, each serving a different purpose in the development cycle.

2. What is the significance of `git merge --no-ff`?

`git merge --no-ff` creates a merge commit even if the merge could be resolved as a fast-forward, preserving the history of a feature branch.

3. How do you revert a Git repository to a previous commit?

Use `git reset --hard <commit-hash>` to revert to a specific commit, discarding all changes since that commit.

4. What is a detached HEAD in Git?

A detached HEAD occurs when you check out a commit, branch, or tag that is not the latest commit of a branch.

5. How do you fix a detached HEAD?

Create a new branch from the detached HEAD state with `git branch <new-branch>` and check it out with `git checkout <new-branch>` to move back to a non-detached state.

6. Explain the difference between `git pull` and `git fetch` followed by `git merge`.

`git pull` does a `git fetch` followed by a `git merge` automatically. Using `git fetch` followed by `git merge` allows you to review changes before merging.

7. What is `git rebase --interactive`?

`git rebase --interactive` allows you to modify commits in many ways, such as rewriting, combining, and removing commits in a more controlled manner.

8. How do you squash the last N commits into a single commit?

Use `git rebase --interactive HEAD~N` and choose `squash` for the commits you want to combine.

9. What are submodules in Git?

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.

10. How do you update a submodule?

Use `git submodule update --remote` to fetch and update your submodules.

11. What is `git bisect`? How do you use it?

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

12. How do you change the URL of a remote repository?

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

13. What is the significance of `git push --force`?

`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.

14. How do you clean untracked files from your working directory?

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

15. What is `git reflog`?

`git reflog` shows a log of where the HEAD and branch references have been, allowing you to navigate back to previous states.

16. How do you resolve a rebase conflict?

Resolve the conflict manually in the affected files, then use `git add` to stage the resolved files, and continue the rebase with `git rebase --continue`.

17. What is a bare repository in Git?

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.

18. How do you rename a remote branch?

Rename the local branch, push it to the remote, and then delete the old remote branch.

19. What is the purpose of `git tag -a`?

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

20. How do you find a list of files that have changed in a specific commit?

Use `git show --name-only <commit-hash>` to list the files that changed in a commit.

21. What is `git blame` and how do you use it?

`git blame` shows what revision and author last modified each line of a file. It's useful for tracking changes and identifying who made them.

22. How do you configure Git to ignore changes in file permissions?

Use `git config core.fileMode false` to ignore file permission changes.

23. What is the difference between `HEAD`, `working tree` and `index` in Git?

`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.

24. How do you make an existing Git branch track a remote branch?

Use `git branch --set-upstream-to=<remote>/<branch> <local-branch>` to set a local branch to track a remote branch.

25. What does `git fetch --prune` do?

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

26. How do you combine multiple commits into one without merging?

Use `git rebase --interactive` to squash commits into a single commit without creating a merge commit.

27. What is `git stash drop`?

`git stash drop` removes a single stashed state from the stash list.

28. How do you list all the remote branches?

Use `git branch -r` to list all remote branches.

29. What is the purpose of `git gc` (garbage collection)?

`git gc` cleans up unnecessary files and optimizes the local repository.

30. How do you find who introduced a line of code using Git?

Use `git blame <file-name>` to see who last modified each line of a file.

31. What does `git commit --dry-run` do?

It simulates a commit, showing what would be committed without actually committing the changes.

32. How do you revert changes made to the working directory?

Use `git checkout -- <file-name>` to discard changes in the working directory.

33. What is the purpose of `git log --graph`?

`git log --graph` displays the commit history in a graphical representation.

34. How do you list all tags in Git?

Use `git tag` to list all tags in the current repository.

35. What is `git show` and how do you use it?

`git show <commit-hash>` displays the information about a git object like a commit.

36. How do you copy a commit from one branch to another?

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

37. What is `git archive`?

`git archive` is used to create an archive (zip or tar) of files from a named tree.

38. What does `git checkout --track <remote/branch>` do?

It creates a new branch that tracks the specified remote branch.

39. How do you compare two branches in Git?

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

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

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

Popular Git Interview Questions and Answers

1. What is Git?

Git is a decentralized system for version control that enables developers to monitor and control modifications to their code repository.

2. How do you clone a repository?

Use `git clone <repository-url>` to make a copy of the target repository on your local machine.

3. `git pull` vs `git fetch`

`git pull` updates your current branch with the latest changes from the remote, while `git fetch` retrieves the latest changes from the remote without integrating them into your local branch.

4. Explain the Git workflow.

The basic Git workflow involves creating branches, making changes, committing those changes, and then merging those changes back into the main branch.

5. How do you fix a merge conflict?

Fix merge conflicts by editing the conflicted files to choose which changes to keep, then staging and committing those changes.

6. What is a branch in Git?

A branch in Git is a separate line of development, allowing you to work on different features or fixes independently.

7. How do you create a new branch and switch to it?

Use `git checkout -b <branch-name>` to create and switch to a new branch.

8. What is a commit in Git?

A commit represents a specific moment's capture of your repository, documenting the modifications made to your project.

9. How do you push changes to a remote repository?

Use `git push <remote-name> <branch-name>` to send your committed changes to a remote repository.

10. What is `git merge` and how do you use it?

`git merge <branch>` merges changes from one branch into the current branch.

11. What is a `.gitignore` file?

A `.gitignore` file identifies files that should remain untracked and be disregarded by Git on purpose.

12. How do you revert a commit?

To create a new commit that reverses the changes introduced in a specific commit, execute the command `git revert <commit-hash>`.

13. What is a fast-forward merge?

A fast-forward merge happens when the target branch's head is behind the merged branch's head, allowing the target branch to fast-forward to the tip of the merged branch.

14. How do you change a commit message that you have already pushed?

Use `git commit --amend` to change the last commit message, then use `git push --force` to update the remote repository.

15. What is the difference between `git checkout`, `git reset`, and `git revert`?

`git checkout` switches branches or restores working tree files, `git reset` changes the head to a specific state, and `git revert` undoes changes by creating a new commit.

16. How do you squash commits?

Use `git rebase -i` and then choose to squash the commits in the interactive prompt.

17. What is `git stash`?

`git stash` temporarily shelves changes so you can work on a different branch with a clean working directory.

18. How do you list all the remote connections for a repository?

Use `git remote -v` to list all remote connections.

19. What does `git fetch` do?

`git fetch` updates your local copy of a remote branch, without merging the changes into your current branch.

20. How do you delete a branch locally and remotely?

Locally: `git branch -d <branch-name>`, remotely: `git push <remote-name> --delete <branch-name>`.

21. What is the purpose of `git config`?

`git config` is used to set configuration options for your Git installation, such as user name and email.

22. How do you list all the branches that are merged into the current branch?

Use `git branch --merged` to list branches merged into the current branch.

23. What is `git log` and how do you use it?

`git log` displays the commit history. You can use various options to format the output, such as `--oneline`, `--graph`, etc.

24. How do you find a specific commit by message?

24. How do you find a specific commit by message?

Use `git log --grep="commit message"` to search the commit history for a specific message.

25. What is the use of `git diff`?

`git diff` shows the differences between commits, commit and working directory, etc.

26. How do you add files to a commit?

Use `git add <file-name>` to stage a file for commit.

27. What does `git commit -m "message"` do?

It commits the staged changes to the repository with a message describing the commit.

28. How do you update a Git repository to the latest version?

Execute the command `git pull` to retrieve and integrate updates from the remote repository into your local branch.

29. What is `git branch -d` and how do you use it?

`git branch -d <branch-name>` deletes a local branch, if it has been fully merged in its upstream branch.

30. How do you see the changes made by a specific commit?

Use `git show <commit-hash>` to display the changes made in a commit.

31. What is the significance of the HEAD pointer in Git?

HEAD points to the most recent commit on the current branch, indicating the workspace's latest status.

32. How do you rename a local Git branch?

Use `git branch -m <old-name> <new-name>` to rename a local branch.

33. What is the purpose of `git checkout -- <file>`?

33. What is the purpose of `git checkout -- <file>`?

It is used to discard changes in the working directory for the specified file.

34. How do you create a tag in Git?

Use `git tag <tag-name>` to create a lightweight tag, or `git tag -a <tag-name> -m "message"` for an annotated tag.

35. What is `git push --tags`?

It pushes all your tags to the remote repository.

36. How do you revert to a previous commit without losing the changes made since?

Use `git revert <commit-hash>` for each commit you want to revert. This creates new commits that reverse the changes.

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

`git rm <file>` removes files from the working directory and stages the deletion.

38. How do you compare two commits?

Use `git diff <commit1> <commit2>` to see the differences between two commits.

39. What is `git rebase` and how is it different from merge?

`git rebase` rewrites the commit history by moving the branch to the tip of the target branch, whereas merge combines two histories together.

40. How do you handle a merge conflict in Git?

Resolve the conflict manually in the affected files, mark them as resolved with `git add`, and then complete the merge with `git commit`.

Learn from Experts in the Industry!

DevOps Engineer Masters Program

EXPLORE PROGRAM

Advanced Git Interview Questions and Answers

1. What is the Git object model?

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).

2. Explain the difference between `git merge` and `git rebase` and when you would use each.

`git merge` integrates changes from one branch to another, preserving the history of both branches. `git rebase` rewrites the history by placing commits from one branch onto another, creating a linear history. Use `merge` to preserve the history of a feature branch, and `rebase` to clean up the commit history before merging.

3. How does Git store data?

Git preserves project data by taking snapshots at various points in time. Every commit acts as a record of the files' condition at that specific instance. To manage this information effectively, Git employs both delta compression and the use of direct object references.

4. What is the role of the `.git/index` file?

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

5. Explain what a "detached HEAD" is and how you might end up in one.

A detached HEAD occurs when you check out a specific commit rather than a branch. This temporary state allows you to navigate through the history of the repository without making changes to the branches.

6. How do you interactively rebase the last N commits?

6. How do you interactively rebase the last N commits?

Use `git rebase -i HEAD~N` where N is the number of commits you want to rebase. This opens an editor showing the last N commits and allows you to reorder, squash, edit, or drop commits.

7. What are the advantages of using a rebase over a merge?

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.

8. What is a git hook and how might you use it?

Git hooks are automated scripts triggered before or after specific Git commands, such as pre-commit, pre-push, post-commit, and post-receive, are executed. They serve various functions, including syntax verification, testing, and the enforcement of project guidelines.

9. How do you squash the last N commits into one using Git?

Use `git rebase -i HEAD~N` and mark all but the first commit with "squash" or "fixup" to combine them into a single commit.

10. What is the Git Garbage Collection and when is it called?

Git's garbage collection (`git gc`) is a housekeeping task that cleans up unnecessary files and optimizes the local repository. It is automatically called by certain commands but can be manually triggered for maintenance.

11. How do you find and restore a deleted file in the Git history?

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.

12. What is the significance of `git push --force-with-lease` over `git push --force`?

`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.

13. How can you achieve a Git bisect manually?

Manually performing a bisect involves checking out commits in a binary search manner to narrow down the commit that introduced a bug, but this is automated with `git bisect`.

14. Explain the difference between `git stash pop` and `git stash apply`.

`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.

15. How do you reapply a commit that has been reverted?

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

16. What is the purpose of `git cherry-pick`?

`git cherry-pick` is used to apply the changes introduced by some commits from one branch onto another branch.

17. How do you resolve a binary file conflict in Git?

Binary file conflicts must be resolved manually by choosing which version of the file to keep, then adding and committing that file.

18. What is the function of `git blame -L`?

`git blame -L` shows who last modified each line of a file within a given line range.

19. How do you handle large files with Git?

Use Git Large File Storage (LFS) to handle large files by storing references in the repository but keeping the actual files on a separate server.

20. What is the use of `git submodule` and how do you update one?

Git submodules enable you to maintain a Git repository as a subdirectory within a different Git repository. To update a submodule, use the command `git submodule update --remote`.

21. How do you list all the aliases you have set in Git?

21. How do you list all the aliases you have set in Git?

Use `git config --get-regexp alias` to list all aliases set in the Git configuration.

22. Explain how to change the base branch of a pull request.

Modifying the base branch of a pull request usually requires altering the target branch through the pull request interface provided by a Git hosting platform (such as GitHub or GitLab).

23. What does `git reflog` show and how can it be useful?

`git reflog` shows a log of the changes to the local repository's HEAD. It can be useful for recovering lost commits or exploring changes made to branches.

24. 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.

25. 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.

26. How do you find which commit a bug was introduced in?

Use `git bisect` to perform a binary search through the commit history to find the commit that introduced a bug.

27. What is the purpose of `git worktree`?

27. What is the purpose of `git worktree`?

`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.

28. How do you configure Git to use a proxy?

Configure Git to use a proxy by setting the `http.proxy` or `https.proxy` configuration, for example, `git config --global http.proxy proxy-url`.

29. What is the difference between a shallow clone and a deep clone in Git?

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.

30. How do you amend the author of a previous commit?

Use `git commit --amend --author="New Author Name <email@domain.com>"` to change the author of the most recent commit, or use an interactive rebase for older commits.

31. What is the purpose of `git fsck` and how do you use it?

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

32. How do you create a patch with Git?

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

33. What is the use of `git revert --no-commit`?

`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.

34. How do you implement a Git workflow in a team environment?

Implement a Git workflow by establishing a branching model (like Git Flow or GitHub Flow), setting up a code review process, defining commit message guidelines, and integrating CI/CD pipelines.

pipelines.

35. What are Git hooks and how can they be customized for a project?

Git hooks are executable scripts triggered by Git prior to or following events like commits, pushes, and receptions. These scripts can be tailored by placing them in the `.git/hooks` directory.

36. How do you troubleshoot connectivity issues when using Git with a remote repository?

Troubleshoot connectivity issues by checking network connections, verifying remote repository URLs, ensuring authentication credentials are correct, and reviewing proxy configurations.

37. What is the significance of the `git fetch --tags` command?

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

38. How can you use Git to track changes in any file, even if it's not code?

Git can track changes in any file type (binary or text) by adding the file to the repository and committing changes. For binary files, it's often beneficial to use Git LFS.

39. Explain the process of contributing to an open-source project using Git.

Contributing to an open-source project typically involves forking the project repository, cloning your fork, making changes in a new branch, pushing the branch to your fork, and submitting a pull request to the original repository.

40. How do you manage multiple configurations for different projects in Git?

Manage multiple configurations by using the `git config` command with the `--global`, `--system`, or `--local` flags to set configuration options at different levels or by using includeIf in the Git configuration to include specific configurations based on the repository path.

Become a DevOps Expert!

Embark on a transformative journey towards mastering the art of DevOps with this Post Graduate Program in DevOps, developed by Simplilearn in partnership with Caltech. This course is designed to equip you with the skills and knowledge required to excel in the fast-paced world of DevOps. Whether you're aiming to streamline project lifecycles, enhance efficiency in deployment, or optimize operational workflows, this program offers a deep dive into the methodologies and tools that are essential for success.

Find our Post Graduate Program in DevOps Online Bootcamp in top cities:

Name	Date	Place
Post Graduate Program in DevOps	Cohort starts on 24th Feb 2024, Weekend batch	Your City
Post Graduate Program in DevOps	Cohort starts on 6th Apr 2024, Weekend batch	Your City

About the Author

Simplilearn

Simplilearn is one of the world's leading providers of online training for Digital Marketing, Cloud Computing, Project Management, Data Science, IT, Software Development, and many other em…

View More

Recommended Programs

Post Graduat

Post Graduate Program in DevOps

7102 Learners

Lifetime Access*

DevOps Engineer

24984 Learners

Lifetime Access*

IIT Guwahati

Professional Certificate Program in Cloud Computing and DevOps

1413 Learners

Lifetime Access*

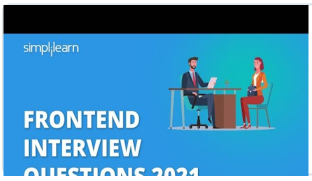
*Lifetime access to high-quality, self-paced e-learning content.

Explore Category

Find Post Graduate Program in DevOps in these cities

Post Graduate Program in DevOps, New Delhi

Recommended Resources



Top 75+ Frontend Developer Interview Qu...

DevOps Interview Guide