DevOps Shack

200 Git Interview Q&A

[Click Here To Enrol To Batch-5 | DevOps & Cloud DevOps]

1. **What is Git?**
    - Git is a distributed version control system used to track changes in source code during software development.
2. **What is a repository in Git?**
    - A repository is a storage location for software packages, which includes the entire history of changes and revisions.
3. **How do you initialize a new Git repository?**
    - Use the command: `git init`.
4. **How do you clone a repository?**
    - Use the command: `git clone <repository-url>`.
5. **What is a commit in Git?**
    - A commit is a snapshot of the project's state at a specific point in time.
6. **How do you create a new branch in Git?**
    - Use the command: `git branch <branch-name>`.
7. **How do you switch to a different branch?**
    - Use the command: `git checkout <branch-name>`.
8. **What is the difference between `git fetch` and `git pull`?**

- `git fetch` downloads changes from the remote repository without integrating them. `git pull` downloads and merges changes.

9. **How do you stage changes for a commit?**
    - Use the command: `git add <file-or-directory>`.
10. **How do you commit staged changes?**
    - Use the command: `git commit -m "commit message"`.

Intermediate Git Questions

11. **What is a merge conflict?**
    - A merge conflict occurs when changes in different branches interfere with each other, requiring manual resolution.
12. **How do you resolve a merge conflict?**
    - Edit the conflicting files to resolve differences and then stage the resolved files with `git add`.
13. **What is a tag in Git?**
    - A tag is a reference to a specific point in the Git history, often used to mark releases.
14. **How do you create a tag?**
    - Use the command: `git tag <tag-name>`.
15. **What is a remote repository?**
    - A remote repository is a version of your project hosted on the internet or network.
16. **How do you add a remote repository?**
    - Use the command: `git remote add <name> <url>`.
17. **How do you view the commit history?**
    - Use the command: `git log`.
18. **How do you undo the last commit?**
    - Use the command: `git reset --soft HEAD~1`.

19. **What is the purpose of a .gitignore file?**
    - A .gitignore file specifies intentionally untracked files that Git should ignore.
20. **How do you remove a file from Git without deleting it from the filesystem?**
    - Use the command: `git rm --cached <file>`.

Advanced Git Questions

21. **What is rebasing in Git?**
    - Rebasing is the process of moving or combining a sequence of commits to a new base commit.
22. **How do you perform an interactive rebase?**
    - Use the command: `git rebase -i <commit>`.
23. **What is a Git hook?**
    - Git hooks are scripts that run automatically on specific Git events.
24. **How do you create a Git hook?**
    - Place an executable script in the `.git/hooks` directory with the name of the hook (e.g., `pre-commit`).
25. **What is the difference between `git reset` and `git revert`?**
    - `git reset` moves the branch pointer backward and can alter commit history. `git revert` creates a new commit that undoes changes.
26. **How do you squash commits?**
    - Use interactive rebase: `git rebase -i <commit>`, and change `pick` to `squash` for the commits you want to combine.
27. **What is `git cherry-pick`?**
    - `git cherry-pick` applies the changes from a specific commit onto the current branch.
28. **How do you apply a patch in Git?**
    - Use the command: `git apply <patch-file>`.
29. **What is `git bisect`?**
    - `git bisect` is a tool used to find the commit that introduced a bug by performing a binary search through the commit history.
30. **How do you use `git bisect`?**

- Start with `git bisect start`, mark a commit as good with `git bisect good <commit>`, and mark a commit as bad with `git bisect bad <commit>`.

Expert Git Questions

31. **What is Git stash?**
    - Git stash temporarily shelves changes in the working directory that are not ready to be committed.
32. **How do you create a stash?**
    - Use the command: `git stash`.
33. **How do you apply a stash?**
    - Use the command: `git stash apply`.
34. **How do you list stashes?**
    - Use the command: `git stash list`.
35. **What is the difference between `git merge` and `git rebase`?**
    - `git merge` combines changes from different branches, creating a merge commit. `git rebase` moves or combines a sequence of commits to a new base.
36. **How do you force push to a remote repository?**
    - Use the command: `git push --force`.
37. **What is `git reflog`?**
    - `git reflog` records changes made to the tip of branches and other references.
38. **How do you view the reflog?**
    - Use the command: `git reflog`.
39. **What is `git fsck`?**
    - `git fsck` verifies the connectivity and validity of objects in the repository.
40. **How do you verify the integrity of a repository?**
    - Use the command: `git fsck`.

More Real-time Scenarios and Solutions

41. **How do you rename a branch?**

- Rename the current branch: `git branch -m new-branch-name`.
- Rename a branch from another branch: `git branch -m old-branch-name new-branch-name`.

42. **How do you delete a remote branch?**
    - Use the command: `git push origin --delete <branch-name>`.
43. **How do you ignore changes to a tracked file?**
    - Use the command: `git update-index --assume-unchanged <file>`.
44. **How do you track changes again to a previously ignored file?**
    - Use the command: `git update-index --no-assume-unchanged <file>`.
45. **What is a bare repository?**
    - A bare repository is a repository that doesn't have a working directory and is typically used as a remote repository.
46. **How do you create a bare repository?**
    - Use the command: `git init --bare`.
47. **How do you export a Git repository to a tar file?**
    - Use the command: `git archive --format=tar --output=<filename>.tar <branch-name>`.
48. **How do you import a repository from a tar file?**
    - Use the command: `tar -xvf <filename>.tar`.
49. **How do you view changes made in a specific commit?**
    - Use the command: `git show <commit-id>`.
50. **How do you undo changes in the working directory?**
    - Use the command: `git checkout -- <file>`.

Additional Intermediate Questions

51. **How do you find a specific commit based on a message?**
    - Use the command: `git log --grep="<message>"`.
52. **How do you set up a global Git configuration?**

- Use the command: `git config --global user.name "Your Name"` and `git config --global user.email "your.email@example.com"`.

53. **How do you set up a Git alias?**
    - Use the command: `git config --global alias.<alias-name> <git-command>`.
54. **What is the difference between `HEAD`, `FETCH_HEAD`, `ORIG_HEAD`, and `MERGE_HEAD`?**
    - `HEAD`: The current commit your working directory points to.
    - `FETCH_HEAD`: The branch you last fetched from.
    - `ORIG_HEAD`: The original head before performing a destructive operation.
    - `MERGE_HEAD`: The commit(s) you're merging in.
55. **How do you reset a file to a specific commit?**
    - Use the command: `git checkout <commit-id> -- <file>`.
56. **How do you compare changes between two branches?**
    - Use the command: `git diff <branch1> <branch2>`.
57. **What is a submodule in Git?**
    - A submodule is a repository embedded inside another repository.
58. **How do you add a submodule?**
    - Use the command: `git submodule add <repository-url>`.
59. **How do you update a submodule?**
    - Use the command: `git submodule update --remote`.
60. **How do you remove a submodule?**
    - Use the commands:
    - ```
      git submodule deinit -f path/to/submodule
      git rm -f path/to/submodule
      ```

Additional Advanced Questions

61

. **How do you create a patch from the last commit?** - Use the command: `git format-patch -1`.

62. **How do you apply a patch file?**
      ```

- Use the command: `git apply <patch-file>`.

63. **How do you create a custom Git command?**
    - Create an executable script named `git-<command-name>` and place it in your PATH.
64. **What is the use of the `git rerere` command?**
    - `git rerere` stands for "reuse recorded resolution" and helps in reusing conflict resolutions.
65. **How do you enable rerere in Git?**
    - Use the command: `git config --global rerere.enabled true`.
66. **How do you archive a specific branch in Git?**
    - Use the command: `git archive --format=tar <branch-name> | gzip > <filename>.tar.gz`.
67. **How do you rename a remote repository?**
    - Use the command: `git remote rename <old-name> <new-name>`.
68. **How do you change the URL of a remote repository?**
    - Use the command: `git remote set-url <remote-name> <new-url>`.
69. **How do you revert a merge commit?**
    - Use the command: `git revert -m 1 <merge-commit-id>`.
70. **How do you reapply a commit that was reverted?**
    - Use the command: `git cherry-pick <reverted-commit-id>`.

Git Best Practices and Patterns

71. **What are Git hooks and why are they useful?**
    - Git hooks are scripts that run automatically on specific Git events. They help enforce policies and automate workflows.
72. **How do you create a pre-commit hook?**
    - Create an executable script named `pre-commit` in the `.git/hooks` directory.
73. **How do you enforce code style with a pre-commit hook?**
    - Add a script in `pre-commit` that runs style checks before allowing commits.

74. **What is Git Flow and how is it used?**
    - Git Flow is a branching model for Git that defines a strict branching strategy designed around the project release.
75. **How do you initialize Git Flow in a repository?**
    - Use the command: `git flow init`.
76. **How do you start a feature in Git Flow?**
    - Use the command: `git flow feature start <feature-name>`.
77. **How do you finish a feature in Git Flow?**
    - Use the command: `git flow feature finish <feature-name>`.
78. **What is the difference between Git Flow and GitHub Flow?**
    - Git Flow involves multiple long-lived branches for development, whereas GitHub Flow uses a simpler workflow with only one long-lived branch (main) and feature branches.
79. **What is the best practice for writing commit messages?**
    - Use short (50 chars or less) summary lines, followed by a blank line and a detailed description.
80. **How do you rebase safely in a shared repository?**
    - Communicate with your team, ensure you are the only one working on the branch, and use `git pull --rebase` instead of `git pull`.

Troubleshooting Git Issues

81. **How do you recover a deleted branch?**
    - Find the commit hash using `git reflog` and create a new branch from it: `git checkout -b <branch-name> <commit-hash>`.
82. **How do you fix a detached HEAD?**
    - Create a new branch or switch back to an existing branch: `git checkout -b <branch-name>` or `git checkout <branch-name>`.
83. **How do you deal with a corrupted Git repository?**
    - Try `git fsck` to find issues and `git reflog` to recover lost commits. Restore from a backup if necessary.

84. **How do you clean up a large repository?**
    - Use `git gc`, `git prune`, and consider using Git LFS for large files.
85. **How do you resolve a merge conflict in a binary file?**
    - Manual intervention is often required. Use tools like `git mergetool` and communicate with team members to decide the correct version.
86. **How do you remove sensitive data from a repository?**
    - Use `git filter-branch` or tools like `BFG Repo-Cleaner`.
87. **How do you troubleshoot slow Git operations?**
    - Check for large files, use `git gc` to clean up the repository, and optimize your network connection.
88. **What is `git fsck` and when would you use it?**
    - `git fsck` verifies the connectivity and validity of objects in the database. Use it to diagnose repository issues.
89. **How do you handle a "bad object" error in Git?**
    - Use `git fsck` to find and fix corrupted objects. Consider restoring from a backup if the issue persists.
90. **How do you manage multiple repositories efficiently?**
    - Use submodules, subtree, or GitHub organizations. Tools like GitLab CI/CD can also help manage multiple projects.

Advanced Git Configuration

91. **How do you set up Git to use a different editor?**
    - Use the command: `git config --global core.editor "<editor>"`.
92. **How do you enable colored output in Git?**
    - Use the command: `git config --global color.ui auto`.
93. **How do you configure Git to handle line endings?**
    - Use the command: `git config --global core.autocrlf true` (for Windows) or `git config --global core.autocrlf input` (for Unix).
94. **How do you set up credential caching in Git?**
    - Use the command: `git config --global credential.helper cache`.

95. **How do you set up a global Git ignore file?**
    - Create a global ignore file and configure it: `git config --global core.excludesfile ~/.gitignore_global`.
96. **How do you sign your commits with GPG?**
    - Generate a GPG key, configure Git to use it: `git config --global user.signingkey <key-id>`, and sign commits with `git commit -S`.
97. **How do you customize the Git prompt?**
    - Modify your shell configuration file (e.g., `.bashrc`, `.zshrc`) to include Git status in the prompt.
98. **How do you use Git aliases to improve productivity?**
    - Create aliases for common commands: `git config --global alias.st status`, `git config --global alias.co checkout`.
99. **How do you configure Git to work behind a proxy?**
    - Use the command: `git config --global http.proxy <proxy-url>`.
100. **How do you configure Git for better performance in large repositories?** - Use sparse-checkout, shallow clone, and `git gc` for maintenance.

More Practical Git Commands

101. **How do you amend a commit message?**
    - Use the command: `git commit --amend -m "New commit message"`.
102. **How do you create a patch from a specific commit range?**
    - Use the command: `git format-patch <start-commit>..<end-commit>`.
103. **How do you revert changes in a specific file to a previous commit?**
    - Use the command: `git checkout <commit-id> -- <file>`.
104. **How do you list all tags in a repository?**
    - Use the command: `git tag`.
105. **How do you show the commit that introduced a specific line in a file?**
    - Use the command: `git blame <file>`.

106. **How do you list all branches in a repository?**
    - Use the command: `git branch`.

107. **How do you delete a local branch?**
    - Use the command: `git branch -d <branch-name>`.

108. **How do you delete a remote branch?**
    - Use the command: `git push origin --delete <branch-name>`.

109. **How do you find the differences between two commits?**
    - Use the command: `git diff <commit1> <commit2>`.

110. **How do you list all remote repositories?**
    - Use the command: `git remote -v`.

More Scenario-Based Questions

111. **How do you handle a situation where you need to merge changes from multiple branches?**
    - Merge each branch into the target branch, resolve conflicts as they arise, and ensure all changes are properly integrated.

112. **How do you synchronize your branch with the latest changes from the main branch?**
    - Use the command: `git pull origin main`.

113. **How do you manage a feature branch when the main branch has diverged?**
    - Use `git rebase` to reapply your changes on top of the latest main branch.

114. **How do you handle a situation where you accidentally committed sensitive information?**
    - Remove the sensitive information using `git filter-branch` or BFG Repo-Cleaner and force-push the changes.

115. **How do you ensure that your commits follow a specific style guide?**
    - Use Git hooks, like a pre-commit hook, to enforce style checks and linting before commits are allowed.

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

- Use Git LFS (Large File Storage) to track large binary files efficiently.

117. **How do you setup continuous integration (CI) with Git?**

- Integrate your repository with a CI tool like Jenkins, GitHub Actions, or GitLab CI, and define your build and test workflows.

118. **How do you handle submodules in a repository?**

- Use commands like `git submodule add`, `git submodule update`, and `git submodule init` to manage submodules.

119. **How do you create a Git alias for a frequently used command?**

- Use the command: `git config --global alias.<alias-name> <git-command>`.

120. **How do you handle a scenario where you need to reapply a series of commits on top of another branch?**

- Use `git rebase` to reapply the series of commits onto the target branch.

Even More Questions

121. **What is the purpose of `git fsck`?**
- `git fsck` checks the integrity of the Git repository and identifies issues.

122. **How do you display a graphical representation of the commit history?**

- Use the command: `git log --graph`.

123. **How do you revert a file to the state of a specific commit?**

- Use the command: `git checkout <commit-id> -- <file>`.

124. **How do you cherry-pick multiple commits?**

```
- Use the command: `git cherry-pick <commit1> <commit2>`.
```

128. **How do you set up a global ignore file in Git?**

- Create the global ignore file and configure Git: `git config --global core.excludesfile <file-path>`.

129. **How do you add all changes in a directory to the staging area?**

- Use the command: `git add <directory>`.

130. **How do you rename a file and keep its history in Git?**

- Use the command: `git mv <old-filename> <new-filename>`.

131. **How do you list the contributors to a repository?**

- Use the command: `git shortlog -s -n`.

132. **How do you create an annotated tag?**

- Use the command: `git tag -a <tag-name> -m "message"`.

133. **How do you list all branches that have been merged into the current branch?**

- Use the command: `git branch --merged`.

Collaborative Workflows

131. **How do you create a pull request?**

- Push your branch to the remote repository and create a pull request via the repository hosting service (e.g., GitHub, GitLab).

132. **How do you fetch and checkout a pull request locally?**

- Use the command: `git fetch origin pull/<ID>/head:<branch-name>`.

133. **How do you update a pull request after making changes?**

- Commit your changes and push to the branch associated with the pull request.

134. **How do you review changes in a pull request?**

- Use the repository hosting service's review tools to comment, approve, or request changes.

135. **How do you resolve conflicts in a pull request?**

- Pull the latest changes, resolve conflicts locally, commit, and push the resolved changes.

136. **How do you ensure your branch is up to date with the main branch before creating a pull request?**

- Use the command: `git pull origin main` and resolve any conflicts.

137. **How do you squash commits before merging a pull request?**
    - Use interactive rebase to squash commits: `git rebase -i <base-commit>`.
138. **How do you close a pull request?**
    - Merge the pull request or close it via the repository hosting service.
139. **How do you assign reviewers to a pull request?**
    - Use the repository hosting service's interface to assign reviewers.
140. **How do you link issues to a pull request?**
    - Mention the issue number in the pull request description (e.g., "Fixes #123").

More Git Best Practices

141. **What are some common Git branching strategies?**
    - Git Flow, GitHub Flow, GitLab Flow, and Trunk-Based Development.
142. **How do you enforce a consistent commit message format?**
    - Use commit message templates and pre-commit hooks.
143. **How do you prevent large files from being added to a repository?**
    - Use a pre-commit hook to check file sizes and reject large files.
144. **How do you manage code reviews in a Git workflow?**
    - Use pull requests and assign reviewers.
145. **How do you handle release management with Git?**
    - Use tags to mark release points and maintain a release branch.
146. **How do you document your Git workflow?**
    - Create a CONTRIBUTING.md file in the repository with guidelines and best practices.
147. **How do you ensure that your repository is clean and optimized?**
    - Regularly run `git gc` and `git prune` to clean up and optimize the repository.
148. **How do you enforce code quality checks in a Git workflow?**

- Integrate CI tools to run automated tests and code quality checks on each commit or pull request.

149. **How do you manage dependencies in a Git repository?**

- Use submodules or a package manager (e.g., npm, pip) and version control the dependency files.

150. **How do you handle hotfixes in a Git workflow?**

- Create a hotfix branch from the main branch, apply the fix, and merge it back into the main branch and the develop branch.

Git Troubleshooting

151. **How do you resolve a "detached HEAD" state?**

- Create a new branch or switch back to an existing branch: `git checkout -b <branch-name>` or `git checkout <branch-name>`.

152. **How do you fix a merge conflict in a rebase?**

```
- Resolve the conflict, stage the changes, and continue the rebase: `git rebase --continue`.
```

154. **How do you recover from accidentally deleting a branch?**

- Use `git reflog` to find the commit hash and create a new branch: `git checkout -b <branch-name> <commit-hash>`.

155. **How do you troubleshoot a slow Git repository?**

```
- Use `git gc` and `git fsck` to clean and verify the repository. Optimize large files with Git LFS.
```

157. **How do you handle a "fatal: refusing to merge unrelated histories" error?**

- Use the `--allow-unrelated-histories` option in the merge command: `git merge <branch> --allow-unrelated-histories`.

158. **How do you handle a "remote: Repository not found" error?**

- Verify the remote URL and your access permissions.

159. **How do you fix a "non-fast-forward" error when pushing?**

- Use `git pull` to integrate remote changes, resolve any conflicts, and then push.

160. **How do you handle a "detected dubious ownership" warning?**
    - Update the ownership or configuration of the repository to match the current user.
161. **How do you fix a "working tree is dirty" error when rebasing?**
    - Stash or commit your changes before starting the rebase.
162. **How do you resolve a "conflict markers" error in a file?**
    - Manually edit the file to resolve the conflict, stage the resolved file, and commit.

More Practical Commands

161. **How do you stage part of a file?**
    - Use the command: `git add -p <file>`.
162. **How do you create a commit template?**
    - Create a template file and configure Git: `git config --global commit.template <template-file>`.
163. **How do you list all contributors to a repository?**
    - Use the command: `git shortlog -s -n`.
164. **How do you search commit messages?**
    - Use the command: `git log --grep="<search-term>"`.
165. **How do you create a branch from a specific commit?**
    - Use the command: `git checkout -b <branch-name> <commit-id>`.
166. **How do you configure a default branch for new repositories?**

```
- Use the command: `git config --global init.defaultBranch <branch-name>`.
```

172. **How do you display the commit history with a specific format?**
    - Use the command: `git log --pretty=format:"%h - %an, %ar : %s"`.
173. **How do you display the differences between the working directory and the last commit?**
    - Use the command: `git diff HEAD`.

174. **How do you create a branch based on a remote branch?**

    - Use the command: `git checkout -b <branch-name> origin/<remote-branch>`.

175. **How do you untrack a file but keep it in the working directory?**

- Use the command: `git rm --cached <file>`.

Scenario-Based Questions

171. **How do you recover a commit that was accidentally removed?**

    - Use `git reflog` to find the commit hash and create a new branch from it.

172. **How do you ensure all your branches are up to date with the remote repository?**

    - Use the command: `git fetch --all`.

173. **How do you handle a scenario where you need to make a temporary fix in production?**

    - Create a hotfix branch, apply the fix, and merge it into the main and develop branches.

174. **How do you ensure your team follows the same branching strategy?**

    - Document the strategy in a CONTRIBUTING.md file and use branch protection rules.

175. **How do you handle a scenario where multiple developers are working on the same feature?**

    - Use feature branches and regularly merge changes from the main branch to keep the feature branch up to date.

176. **How do you handle a scenario where a critical bug is found just before a release?**

    - Create a hotfix branch, apply the fix, and merge it back into the main and develop branches.

177. **How do you handle a scenario where you need to roll back to a previous release?**

- Use `git checkout` to switch to the commit or tag of the previous release and create a new branch if necessary.

178. **How do you handle a scenario where your repository is growing too large?**
    - Use Git LFS for large files, prune unnecessary branches, and regularly run `git gc`.

179. **How do you handle a scenario where you need to split a monolithic repository into multiple repositories?**
    - Use `git subtree` or `git filter-branch` to split the repository into smaller ones.

180. **How do you handle a scenario where you need to keep certain files in sync across multiple repositories?**
    - Use Git submodules or a shared repository for common files.

More Git Best Practices

181. **How do you ensure commit messages are consistent across the team?**
    - Use commit message templates and pre-commit hooks.

182. **How do you automate code quality checks in a Git workflow?**
    - Integrate CI tools to run automated tests and code quality checks on each commit or pull request.

183. **How do you manage environment-specific configurations in a Git repository?**
    - Use environment-specific configuration files and a .gitignore file to exclude sensitive data.

184. **How do you handle a scenario where you need to maintain multiple versions of a project?**
    - Use release branches and tags to manage different versions of the project.

185. **How do you ensure that sensitive information is not committed to the repository?**
    - Use a pre-commit hook to check for sensitive data and a .gitignore file to exclude it.

186. **How do you handle a scenario where you need to collaborate with external contributors?**

    - Use a fork and pull request workflow to manage contributions from external collaborators.

187. **How do you ensure your repository is clean and optimized?**

    - Regularly run `git gc` and `git prune` to clean up and optimize the repository.

188. **How do you manage large binary files in a Git repository?**

    - Use Git LFS (Large File Storage) to track large binary files efficiently.

189. **How do you enforce code quality standards in a Git workflow?**

    - Use pre-commit hooks, code reviews, and CI tools to enforce code quality standards.

190. **How do you handle a scenario where you need to integrate changes from multiple branches?**

    - Use `git merge` or `git rebase` to integrate changes and resolve any conflicts.

Advanced Troubleshooting

191. **How do you fix a corrupted Git repository?**

    - Use `git fsck` to identify issues, restore from a backup, or re-clone the repository if necessary.

192. **How do you handle a scenario where a merge conflict cannot be resolved?**

    - Revert to a previous commit, reapply the changes, and try the merge again.

193. **How do you recover from a failed rebase?**

    - Use `git rebase --abort` to cancel the rebase and return to the original state.

194. **How do you handle a scenario where a commit is missing from the history?**

```
- Use `git reflog` to find the missing commit and create a new branch from it.
```

198. **How do you troubleshoot a slow Git repository?** - Check for large files, run `git gc`, and optimize your network connection.

199. **How do you handle a scenario where you need to backport a fix to an older version?**

- Create a branch from the older version, apply the fix, and merge it back into the main branch.

200. **How do you handle a scenario where a branch is accidentally deleted?**

- Use `git reflog` to find the commit hash and create a new branch from it.

201. **How do you troubleshoot a scenario where Git operations are taking too long?**

- Check for large files, use shallow clones, and optimize your network connection.

202. **How do you handle a scenario where you need to rename a branch in a remote repository?**

- Rename the branch locally and push the changes to the remote repository: `git branch -m old-branch new-branch`, `git push origin new-branch`, `git push origin --delete old-branch`.

203. **How do you ensure that your Git workflow is scalable and maintainable?**

- Use best practices, regular maintenance, and tools like Git LFS, CI/CD integration, and automated code quality checks.