Devops Shack

200 INTERVIEW QUESTIONS & ANSWERS

devopsshack.com

Ebook 2025

DevOps Shack

200 GIT Interview Questions and Answers

What is Git? Why is it used?

- Git is a distributed version control system used to track changes in source code during software development. It allows multiple developers to collaborate, maintain code history, and revert to earlier versions when necessary.

2. **How do you initialize a Git repository?**
    - Use the command git init in the directory you want to initialize. This creates a .git directory containing all necessary files for version control.
    - Example:

```
mkdir my_project
cd my_project
git init
```

Output: Initialized empty Git repository in /path/to/my_project/.git/

Git Branching

3. **What is a branch in Git, and why is it important?**
    - A branch in Git is a lightweight pointer to a specific commit. Branching allows developers to work on features or bug fixes independently without affecting the main codebase.
4. **How do you create and switch to a new branch?**
    - Command:

```
git checkout -b new-feature
```

    - This creates a branch named new-feature and switches to it.

Git Merging

5. **What are the different types of Git merges?**
    - **Fast-Forward Merge:** Occurs when there's no divergence in the branches.
    - **Three-Way Merge:** Happens when there's divergence, and Git creates a new merge commit.
6. **How do you resolve merge conflicts?**
    - Example Scenario: Two developers modify the same line in a file. Steps:

1. Identify conflicting files:

```
git status
```

2. Open the file, and locate conflict markers:

```
<<<<<<< HEAD
Your changes
=======
Their changes
>>>>>>> branch-name
```

3. Edit the file to keep the desired changes.
4. Mark the conflict as resolved:

```
git add conflicted-file
```

5. Commit the merge:

```
git commit
```

Git Rebase

7. **What is Git Rebase, and how does it differ from Git Merge?**
   - **Git Rebase:** Reapplies commits from one branch onto another. It results in a linear history.
   - **Git Merge:** Combines branches and retains the commit history.
8. **When should you use rebase instead of merge?**
   - Use rebase for maintaining a clean, linear commit history in private branches.
   - Avoid rebasing shared branches to prevent rewriting history.

Git Workflow

9. **What is the Gitflow Workflow?**
   - Gitflow is a branching model that uses feature, release, hotfix, and develop branches.
   - Commands:
     1. Start a feature:

```
git checkout -b feature/my-feature develop
```

     2. Merge the feature:

```
git checkout develop
```

```
git merge feature/my-feature
```

10. **Describe a scenario where you would use git cherry-pick.**
    - Scenario: A bug fix commit needs to be applied to multiple branches.

```
git cherry-pick <commit-hash>
```

Git Collaboration

11. **How do you handle pull requests in Git?**
    - Steps:
        1. Create a pull request (PR) on the platform (e.g., GitHub).
        2. Review the changes, add comments, and approve or request changes.
        3. Merge the PR once approved:

```
git merge <branch-name>
```

Git Configuration

12. **How do you set up a global Git configuration?**
    - Commands:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

Git Stash and Clean

13. **What is Git Stash, and how is it used?**
    - Git Stash saves your uncommitted changes temporarily and allows you to apply them later.
    - Commands:
        1. Save changes:

```
git stash
```

        2. Apply stash:

```
git stash apply
```

Git Advanced Topics

14. **How does Git handle large files?**
    - Use Git Large File Storage (LFS) for tracking large files like binaries.
15. **How do you perform an interactive rebase?**
    - Example:

```
git rebase -i HEAD~3
```

This opens an editor to modify the last three commits.

Git Troubleshooting Scenarios

16. **Scenario: What if you commit sensitive data accidentally?**
    - Steps to remove sensitive data:

```
git filter-branch --force --index-filter \
'git rm --cached --ignore-unmatch <file-path>' \
--prune-empty --tag-name-filter cat -- --all
git push origin --force --all
```

17. **Scenario: You accidentally deleted a branch. How do you recover it?**
    - If the branch was merged:

```
git checkout -b branch-name <commit-hash>
```

18. **Scenario: How do you handle detached HEAD?**
    - Commands:

```
git checkout -b new-branch
```

Git Troubleshooting Scenarios (continued)

19. **Scenario: How do you recover a deleted file in Git?**

- If the file was deleted in the working directory but exists in a previous commit:

```
git checkout HEAD -- <file-path>
```

- If it was deleted and committed:
    1. Find the commit:

```
git log -- <file-path>
```

    2. Restore the file:

```
git checkout <commit-hash> -- <file-path>
```

20. **Scenario: How do you revert a commit that was already pushed to the remote?**
    - Use git revert to create a new commit that undoes the changes:

```
git revert <commit-hash>
```

- Push the changes:

```
git push origin <branch-name>
```

21. **What is the difference between git pull and git fetch?**
    - **git fetch:** Downloads changes from the remote repository but does not merge them into your local branch.
    - **git pull:** Combines git fetch and git merge, downloading changes and merging them into your branch.
22. **What is the purpose of the .gitignore file?**
    - .gitignore specifies intentionally untracked files that Git should ignore.
    - Example .gitignore file:

```
# Ignore all .log files
*.log
# Ignore node_modules directory
node_modules/
```

23. **How do you stage and commit changes in Git?**
    - Stage changes:

```
git add <file-path>
```

    - Commit changes:

```
git commit -m "Your commit message"
```

24. **How do you delete a branch locally and remotely?**
    - **Locally:**

```
git branch -d branch-name
```

    - **Remotely:**

```
git push origin --delete branch-name
```

25. **Scenario: You need to rename a branch. How do you do it?**
    - **On local:**

```
git branch -m old-branch-name new-branch-name
```

    - **On remote:**
        1. Delete the old branch:

```
git push origin --delete old-branch-name
```

        2. Push the new branch:

```
git push origin new-branch-name
```

Git Merging

26. **Scenario: How do you avoid automatic merges for specific files?**
    - Use .gitattributes to enforce merge strategies:

*.config merge=ours
Configure:
bash
Copy code
git config --global merge.ours.driver true

27. **Scenario: How do you abort a merge in progress?**
    - Run:

git merge --abort

Git Rebase (continued)

28. **Scenario: How do you squash multiple commits into one?**
    - Use interactive rebase:

git rebase -i HEAD~n

Replace pick with squash or s for commits you want to merge.

29. **Scenario: What happens if a rebase fails?**
    - Resolve conflicts as prompted.
    - Continue the rebase:

git rebase --continue

    - Abort if necessary:

git rebase --abort

Git Workflow (continued)

30. **What is a detached HEAD in Git, and how do you fix it?**
    - A detached HEAD occurs when you checkout a commit directly instead of a branch.
    - Fix by creating a new branch:

git checkout -b new-branch-name

31. **How do you tag a commit and push the tag to a remote?**
    - Tagging:

```
git tag -a v1.0 -m "Version 1.0"
```

- Pushing:

```
git push origin v1.0
```

32. **Scenario: How do you handle a hotfix in production using Gitflow?**
    - Steps:
        1. Create a hotfix branch:

```
git checkout -b hotfix/fix-bug master
```

        2. Apply and commit changes.
        3. Merge back to master and develop:

```
git checkout master
git merge hotfix/fix-bug
git checkout develop
git merge hotfix/fix-bug
```

Git Collaboration (continued)

33. **Scenario: What if someone force-pushed to a shared branch?**
    - Fetch and reset to a known good state:

```
git fetch origin
```

```
git reset --hard origin/branch-name
```

34. **How do you review and test a PR locally?**
    - Fetch the PR:

```
git fetch origin pull/<PR-number>/head:<local-branch-name>
```

    - Check out the branch:

```
git checkout <local-branch-name>
```

35. **Scenario: How do you globally ignore files in Git?**
    - Add patterns to a global ignore file:

```
git config --global core.excludesfile ~/.gitignore_global
```

    - Example ~/.gitignore_global:

```
.DS_Store
*.bak
```

36. **Scenario: How do you enable colorized output in Git?**
    - Command:

```
git config --global color.ui auto
```

37. **Scenario: What if you accidentally dropped a stash?**
    - Recover using the reflog:

```
git reflog
git stash apply stash@{<index>}
```

38. **How do you clean untracked files?**
    - Dry run to preview:

```
git clean -n
```

    - To clean:

```
git clean -f
```

39. **Scenario: How do you clone a specific branch?**
    - Command:

```
git clone -b branch-name --single-branch <repository-url>
```

40. **How do you configure Git hooks?**
    - Create a hook script in .git/hooks/. Example for pre-commit:

```
#!/bin/sh
echo "Running pre-commit hook"
```

41. **Scenario: What if you accidentally committed to the wrong branch?**
    - Steps to move the commit to the correct branch:
        1. Create a new branch:

```
git checkout -b correct-branch
```

        2. Cherry-pick the commit:

```
hgit cherry-pick <commit-hash>
```

        3. Remove the commit from the wrong branch:

```
git checkout wrong-branch
```

```
git reset --hard HEAD~1
```

42. **Scenario: How do you deal with a corrupted Git repository?**
    - Steps:
        1. Run git fsck to diagnose:

```
git fsck
```

        2. If corrupted, reclone the repository:

```
git clone <repository-url>
```

43. **Scenario: How do you identify the author of a specific line in a file?**
    - Use git blame:

```
git blame <file-path>
```

44. **Scenario: What do you do if you accidentally overwrite local changes with a git pull?**
    - Recover changes with git reflog:

```
git reflog
```

```
git checkout <commit-hash>
```

45. **What is the difference between git clone and git fork?**
    - **git clone:** Creates a local copy of a remote repository.
    - **git fork:** Duplicates a repository on platforms like GitHub, creating a separate copy under your account.
46. **What are the different states in Git?**
    - **Untracked:** Files not tracked by Git.
    - **Staged:** Files added to the staging area.
    - **Committed:** Changes saved to the repository.
47. **How do you view the commit history in Git?**
    - Basic log:

```
git log
```

    - One-line summary:

```
git log --oneline
```

48. **How do you undo the last commit?**
    - **Without removing changes:**

```
git reset --soft HEAD~1
```

    - **With removing changes:**

```
git reset --hard HEAD~1
```

49. **What is the difference between git branch and git checkout?**
    - **git branch:** Creates, lists, or deletes branches.
    - **git checkout:** Switches branches or checks out files.
50. **Scenario: How do you force delete a branch that has unmerged changes?**

- Command:

```
git branch -D branch-name
```

51. **What is the difference between merge conflict markers and resolved files?**
    - **Merge conflict markers**:

```
<<<<<<< HEAD
Code from current branch
=======
Code from other branch
>>>>>>> branch-name
```

    - **Resolved files:** After editing the conflict markers and staging the file, it is considered resolved.

52. **Scenario: How do you ensure a branch is up to date before merging?**
    - Steps:
        1. Fetch latest changes:

```
git fetch origin
```

        2. Rebase onto the target branch:

```
git rebase origin/branch-name
```

Git Rebase (continued)

53. **What is an "interactive rebase," and why is it useful?**
    - Interactive rebase allows you to edit, reorder, or squash commits before applying them.
    - Command:

```
git rebase -i HEAD~n
```

    - Use cases:
        - Cleaning up commit history before pushing.
        - Merging related commits into one.

54. **Scenario: How do you handle rebase conflicts?**
    - Steps:

1. Resolve the conflict in the affected files.
2. Stage the resolved files:

```
git add <file>
```

3. Continue the rebase:

```
git rebase –continue
```

55. **Scenario: How do you switch branches and keep your current work?**
    - Use git stash to save changes:

```
git stash
git checkout new-branch
git stash apply
```

56. **What are Git hooks, and how are they used in workflows?**
    - Git hooks are scripts that execute at specific events like commits or merges.
    - Example: Pre-commit hook:
        - Create .git/hooks/pre-commit and add:

```
#!/bin/sh
echo "Checking code quality..."
```

57. **Scenario: How do you pull changes from a specific branch of a remote repository?**
    - Command:

```
git pull origin branch-name
```

58. **How do you set up a remote repository?**
    - Add a remote:

```
git remote add origin <repository-url>
```

    - Push the repository:

```
git push -u origin main
```

59. **What is a Git upstream branch?**
    - An upstream branch is the branch your local branch tracks.
      Example:

```
git branch --set-upstream-to=origin/main main
```

60. **Scenario: How do you configure aliases in Git?**
    - Example:

```
git config --global alias.co checkout
git config --global alias.br branch
```

    - Usage:

```
git co <branch-name>
```

```
git br
```

61. **How do you list all stashes?**
    - Command:

```
git stash list
```

62. **Scenario: How do you drop a specific stash?**
    - Command:

```
git stash drop stash@{index}
```

63. **Scenario: How do you perform a shallow clone?**
    - Command:

```
git clone --depth 1 <repository-url>
```

64. **Scenario: What is Git bisect, and how is it used?**
    - Git bisect helps find the commit that introduced a bug.
    - Steps:
        1. Start bisect:

```
git bisect start
git bisect bad
git bisect good <commit-hash>
```

        2. Test commits and mark them:

```
git bisect good/bad
```

        3. End bisect:

```
git bisect reset
```

65. **Scenario: How do you view the contents of a previous commit?**
    - Command:

```
git show <commit-hash>
```

66. **Scenario: What do you do if you accidentally commit sensitive information (e.g., API keys)?**
    - Steps to remove sensitive data:
        1. Remove the file from all commits:

```
git filter-branch --force --index-filter \
'git rm --cached --ignore-unmatch <file-path>' \
--prune-empty --tag-name-filter cat -- --all
```

        2. Remove the file from the remote:

```
git push origin --force
```

3. Add the file to .gitignore to prevent future issues.

67. **Scenario: What do you do if a git push fails because of a non-fast-forward update?**
    - This happens when the remote has commits that your local branch does not.
    - Steps:
        1. Pull changes:

```
git pull origin branch-name
```

        2. Resolve conflicts if any.
        3. Push again:

```
git push origin branch-name
```

68. **Scenario: How do you recover from an accidental git reset --hard?**
    - Use the reflog to find the lost commits:

```
git reflog
```

    - Checkout the commit:

```
git checkout <commit-hash>
```

69. **What is the difference between git log and git reflog?**
    - **git log:** Shows the commit history of a branch.
    - **git reflog:** Shows a history of all actions performed in the repository (including resets, checkouts).
70. **How do you rename a file in Git?**
    - Rename the file:

```
git mv old-filename new-filename
```

    - Commit the change:

```
git commit -m "Renamed file from old-filename to new-filename"
```

71. **Scenario: How do you create a branch from a specific commit?**
    - Command:

```
git checkout -b new-branch-name <commit-hash>
```

72. **Scenario: How do you list all remote branches?**
    - Command:

```
git branch -r
```

73. **What is the difference between git branch -d and git branch -D?**

- **git branch -d:** Deletes the branch only if it has been merged.
- **git branch -D:** Force deletes the branch even if it has not been merged.

74. **Scenario: How do you prevent Git from creating a merge commit during git pull?**
    - Use the --rebase flag:

```
git pull --rebase
```

75. **What are merge strategies in Git?**
    - **Recursive:** Default strategy for merging two branches.
    - **Ours:** Keeps changes from the current branch.
    - **Octopus:** Used for merging more than two branches.
76. **Scenario: How do you edit a commit message after rebasing?**
    - Use interactive rebase:

```
git rebase -i HEAD~n
```

    - Replace pick with reword for the desired commit.
77. **Scenario: How do you abandon a rebase in progress?**
    - Command:

```
git rebase --abort
```

78. **Scenario: How do you integrate a feature branch into the main branch?**
    - Steps:
        1. Switch to the main branch:

```
git checkout main
```

        2. Merge the feature branch:

```
git merge feature-branch
```

79. **Scenario: What if you want to merge only specific commits from another branch?**
    - Use git cherry-pick:

```
git cherry-pick <commit-hash>
```

80. **What is the purpose of the git reset command?**
    - **Soft Reset:** Moves the HEAD pointer but keeps changes staged.
    - **Mixed Reset (default):** Moves the HEAD pointer and un-stages changes.
    - **Hard Reset:** Moves the HEAD pointer and deletes changes in the working directory.
81. **Scenario: How do you contribute to an open-source repository?**
    - Steps:

1. Fork the repository.
2. Clone the forked repository:

```
git clone <forked-repo-url>
```

3. Create a feature branch:

```
git checkout -b new-feature
```

4. Make changes, commit, and push:

```
git push origin new-feature
```

5. Create a pull request from the forked repo to the original repo.

82. **How do you check the differences between your branch and a remote branch?**
    - Command:

```
git diff branch-name origin/branch-name
```

Git Configuration (continued)

83. **Scenario: How do you configure Git to sign commits with GPG?**
    - Steps:
        1. Generate a GPG key:

```
gpg --gen-key
```

        2. Add the key to Git:

```
git config --global user.signingkey <key-id>
```

        3. Sign commits:

```
git commit -S -m "Signed commit"
```

84. **Scenario: How do you stash only certain files?**
    - Use the -- flag:

```
git stash push <file-path>
```

85. **How do you apply a stash and drop it in one command?**
    - Command:

```
git stash pop
```

86. **Scenario: How do you create and track a Git submodule?**
    - Add the submodule:

```
git submodule add <repository-url> <path>
```

    - Initialize the submodule:

```
git submodule init
```

- Update the submodule:

```
git submodule update
```

87. **What is Git sparse checkout?**
    - Sparse checkout allows you to check out only specific files or directories from a repository.
    - Enable sparse checkout:

```
git sparse-checkout init
git sparse-checkout set <directory>
```

88. **Scenario: How do you resolve conflicts in binary files?**
    - Use an external merge tool configured with Git:

```
git mergetool
```

89. **Scenario: How do you handle a history rewrite on a shared branch?**
    - Communicate with the team and force-push:

```
git push --force
```

90. **Scenario: How do you fix a detached HEAD state?**
    - Create a new branch from the current state:

```
git checkout -b new-branch
```

91. **Scenario: How do you undo a pushed commit without removing it from history?**
    - Use git revert to create a new commit that undoes the changes:

```
git revert <commit-hash>
```

    - Push the revert commit:

```
git push origin branch-name
```

92. **Scenario: What if your git pull results in many unnecessary merge commits?**
    - Use git pull with the --rebase flag to maintain a linear history:

```
git pull --rebase
```

93. **Scenario: How do you troubleshoot when your changes are not visible after pushing?**
    - Possible causes and solutions:
        1. **You pushed to the wrong branch:**

```
git branch
git push origin correct-branch
```

        2. **Remote tracking branch not updated:**

```
git fetch origin
```

94. **Scenario: How do you resolve "detected a conflict during cherry-pick"?**
    - Steps:
        1. Resolve the conflict manually in the affected files.
        2. Stage the resolved files:

```
git add <file>
```

        3. Continue the cherry-pick:

```
git cherry-pick --continue
```

95. **What are lightweight and annotated tags in Git?**
    - **Lightweight tags:** Simple pointers to a specific commit, without additional metadata.

```
git tag lightweight-tag
```

    - **Annotated tags:** Store metadata such as the tagger's name and a message.

```
git tag -a annotated-tag -m "This is an annotated tag"
```

96. **How do you delete a tag?**
    - Locally:

```
git tag -d tag-name
```

    - Remotely:

```
git push origin --delete tag-name
```

97. **What is the difference between HEAD and ORIG_HEAD?**
    - **HEAD:** Refers to the current commit or branch.
    - **ORIG_HEAD:** Refers to the previous state of HEAD, usually before a destructive action like a reset or rebase.

Git Branching (continued)

98. **Scenario: How do you list all branches, including remote and merged branches?**
    - Command:

```
git branch -a  # Lists all branches
git branch --merged  # Lists merged branches
```

99. **Scenario: How do you ensure a branch is based on the latest commit of another branch?**
    - Steps:

1. Switch to your branch:

```
git checkout feature-branch
```

2. Rebase it onto the target branch:

```
git rebase main
```

100. **Scenario: How do you track a remote branch locally?**
- Command:

```
git checkout --track origin/branch-name
```

101. **What is the difference between git merge --squash and git merge --no-ff?**

--squash: Combines all commits into a single commit without creating a merge commit.

--no-ff: Creates a merge commit even if a fast-forward merge is possible.

102. **Scenario: How do you resolve a merge conflict when Git cannot determine which branch has the correct change?** - Open the conflicting file. - Edit manually to retain the correct changes. - Stage the resolved file:

```
git add <file>
diff
```

- Complete the merge:

```
git commit
```

103. **Scenario: How do you rebase a branch while ignoring certain commits?** - Use an interactive rebase:

```
git rebase -i branch-name
```

- Mark commits to be ignored as `drop`.

104. **Scenario: What if a rebase rewrites history you need to recover?** - Use the reflog to recover the previous state:

```
git reflog
```

```
git checkout <previous-commit-hash>
```

105. **Scenario: How do you handle multiple developers working on the same file?** - Best practices:

1. Pull frequently to minimize conflicts:

```
git pull origin branch-name
```

2. Communicate changes to the team.

3. Resolve conflicts collaboratively during merge or rebase.

106. **Scenario: How do you create a temporary branch to test a specific feature?** - Command:

```
git checkout -b temp-branch <commit-hash>
```

107. **Scenario: How do you fetch and check out a pull request from GitHub?** - Command:

```
git fetch origin pull/<PR-number>/head:<local-branch-name>
```

```
git checkout <local-branch-name>
```

108. **Scenario: What do you do if you accidentally push sensitive information to a public repository?**

Steps:

1. Remove the sensitive information:

```
git filter-branch --force --index-filter 'git rm --cached <file>' --prune-empty -- --all
```

2. Force-push the changes:

```
git push origin --force --all
```

109. **Scenario: How do you configure Git to use a custom diff tool?** - Command:

```
git config --global diff.tool custom-diff-tool
```

```
git config --global difftool.custom-diff-tool.path /path/to/tool
```

110. **Scenario: How do you stash changes but keep them in the working directory?** - Use the --keep-index flag:

```
git stash push --keep-index
```

111. **How do you drop all stashes?** - Command:

```
git stash clear
```

112. **What is Git reflog, and when is it used?** - **Reflog:** A log of all references updated in your repository, used for recovering lost commits or branches. - Example:

```
git reflog
```

```
git checkout <commit-hash>
```

113. **Scenario: How do you split a commit into two smaller commits?** - Steps:

1. Reset the commit:

```
git reset HEAD~1
```

2. Stage only part of the changes:

```
git add -p
```

```
git commit -m "First part of the split"
```

3. Commit the remaining changes:

```
git add .
```

```
git commit -m "Second part of the split"
```

114. **Scenario: How do you handle a large repository with too many files?** - Use sparse checkout to limit the files:

```
git sparse-checkout set <path-to-directory>
```

Git Troubleshooting Scenarios (continued)

115. **Scenario: What do you do if your .gitignore changes are not working?** - Possible causes and solutions:

    1. The files are already tracked by Git. Remove them:

```
git rm --cached <file>
```

    2. Double-check .gitignore syntax for errors.

Git Troubleshooting Scenarios (continued)

116. **Scenario: How do you recover a deleted remote branch?**

- If the branch was deleted from the remote repository, you can recover it locally (provided you have a local copy) and push it back:

    1. Verify if the branch exists locally:

```
git branch
```

    2. Push the branch back to the remote repository:

```
git push origin branch-name
```

- If the branch is deleted locally but exists in the remote repository, simply fetch it:

```
git fetch origin branch-name
```

```
git checkout branch-name
```

117. **Scenario: How do you fix a detached HEAD if you accidentally checked out a commit?**

- A detached HEAD occurs when you check out a commit directly instead of a branch. To fix this:

    1. Create a new branch from the detached HEAD:

```
git checkout -b new-branch
```

    2. Alternatively, reattach the HEAD to a branch:

```
git checkout branch-name
```

118. **Scenario: What if your working directory becomes corrupted?**

- Steps to fix:

    1. Identify corrupted files using:

```
git fsck
```

2. Restore the corrupted file from the last good commit:

```
git checkout HEAD -- <file-path>
```

3. If the repository is beyond repair, reclone it:

```
git clone <repository-url>
```

119. **Scenario: How do you recover uncommitted changes after a git stash drop?**

- Use the reflog to find the dropped stash:

1. Check the stash reflog:

```
git reflog
```

2. Locate the stash reference and apply it:

```
git stash apply stash@{<index>}
```

120. **How do you undo changes to a tracked file in your working directory?**

- If the file is tracked and has been modified but not staged, use:

```
git checkout -- <file>
```

- If the file is staged:

```
git reset HEAD <file>
git checkout -- <file>
```

121. **What is the significance of the .git directory in a Git repository?**

- The .git directory contains all the metadata and object data for your repository:
  - **HEAD:** Points to the current branch.
  - **index:** Staging area for changes.
  - **objects:** Stores the content of all commits.
  - **refs:** Contains references to branches and tags.

122. **Scenario: How do you work on multiple features simultaneously in Git?**

- Use separate branches for each feature:

1. Create and switch to a new branch for the first feature:

```
git checkout -b feature-one
```

2. Once done, switch back to the main branch:

```
git checkout main
```

3. Start another feature on a new branch:

```
git checkout -b feature-two
```

123. **Scenario: What do you do if you forget to create a branch before making changes?**

- Save the current state as a new branch:

```
git checkout -b new-branch
```

124. **What is the purpose of git checkout -b?**

- The -b flag creates and switches to a new branch in one command:

```
git checkout -b branch-name
```

125. **Scenario: How do you merge a feature branch back into the main branch after testing?**

- Steps:
    1. Ensure you are on the main branch:

```
git checkout main
```

    2. Merge the feature branch:

```
git merge feature-branch
```

    3. Delete the feature branch if no longer needed:

```
git branch -d feature-branch
```

126. **What is a conflict during merging, and why does it occur?**

- A merge conflict occurs when changes in two branches affect the same lines of a file, and Git cannot determine which changes to keep. This requires manual resolution.

127. **Scenario: How do you handle conflicts during a rebase operation?**

- Steps:
    1. Resolve conflicts in the conflicting files manually.
    2. Stage the resolved files:

```
git add <file>
```

    3. Continue the rebase:

```
git rebase --continue
```

    4. Abort the rebase if necessary:

```
git rebase --abort
```

128. **What are the benefits of rebasing over merging?**

- Rebasing creates a linear, cleaner commit history by avoiding merge commits.
- It makes it easier to navigate through history using git log or git blame.

129. **Scenario: How do you create a patch from a commit?**

- Use git format-patch to create a patch file:

```
git format-patch -1 <commit-hash>
```

- Apply the patch:

```
git apply <patch-file>
```

130. **Scenario: How do you squash multiple commits into one before pushing?**

- Use interactive rebase:
    1. Start the rebase for the last n commits:

```
git rebase -i HEAD~n
```

    2. Replace pick with squash for the commits to squash.
    3. Save and exit the editor.

131. **Scenario: How do you handle changes in a forked repository?**

- Steps:
    1. Add the upstream repository:

```
git remote add upstream <original-repo-url>
```

    2. Fetch the upstream changes:

```
git fetch upstream
```

    3. Merge the upstream changes into your branch:

```
git merge upstream/main
```

132. **How do you inspect changes made by another developer?**

- View the log of their commits:

```
git log --author="developer-name"
```

- See their changes:

```
git diff <commit-hash>
```

133. **Scenario: How do you set a global .gitignore file for all repositories?**

- Steps:
    1. Create a global ignore file:

```
touch ~/.gitignore_global
```

    2. Configure Git to use it:

```
git config --global core.excludesfile ~/.gitignore_global
```

134. **Scenario: How do you stash changes and switch branches safely?**

- Steps:
    1. Stash the changes:

```
git stash
```

    2. Switch branches:

```
git checkout branch-name
```

    3. Reapply the stashed changes:

```
git stash pop
```

135. **What is the difference between git stash apply and git stash pop?**

- **git stash apply:** Reapplies the stash but keeps it in the stash list.
- **git stash pop:** Reapplies the stash and removes it from the stash list.

136. **What is Git rerere, and how is it useful?**

- Git rerere (reuse recorded resolution) remembers how you resolved a conflict so it can automatically resolve the same conflict in the future.
- Enable rerere:

```
git config --global rerere.enabled true
```

137. **Scenario: How do you view changes in a submodule?**

- Use:

```
git diff –submodule
```

138. **Scenario: How do you resolve the error "fatal: refusing to merge unrelated histories"?**

- This error occurs when merging two branches that do not share a common commit history.
- Solution:
    1. Use the --allow-unrelated-histories flag:

```
git merge branch-name --allow-unrelated-histories
```

139. **Scenario: What do you do if Git shows the error "Your local changes would be overwritten by merge"?**

- This error happens when you have uncommitted changes that conflict with the changes being pulled or merged.
- Solution:

1. Stash your changes:

```
git stash
```

2. Perform the merge or pull:

```
git pull origin branch-name
```

3. Reapply your changes:

```
git stash pop
```

140. **Scenario: How do you recover a branch that was accidentally deleted locally and remotely?**

- Steps:

1. Check the reflog for the branch's last commit:

```
git reflog
```

2. Create a new branch from the commit:

```
git checkout -b branch-name <commit-hash>
```

3. Push the branch back to the remote repository:

```
git push origin branch-name
```

141. **Scenario: What do you do if a git fetch or git pull is stuck?**

- Possible solutions:

1. Check your network connection.
2. Add the --verbose flag to debug:

```
git fetch --verbose
```

3. Use shallow fetch to minimize data transfer:

```
git fetch --depth=1
```

142. **What is the purpose of git ls-tree?**

- git ls-tree is used to view the contents of a tree object (e.g., a commit or branch). It lists files and directories along with their types and SHA-1 hashes.
- Example:

```
git ls-tree HEAD
```

143. **How do you compare two commits in Git?**

- Use the git diff command with the two commit hashes:

```
git diff <commit1> <commit2>
```

144. **How do you find a specific file in the commit history?**

- Use git log with the file name:

```
git log -- <file-path>
```

145. **What is the purpose of git archive?**

- git archive is used to create a tar or zip archive of a repository at a specific commit or branch.
- Example:

```
git archive --format=zip HEAD > archive.zip
```

146. **Scenario: How do you set up a branch to track a remote branch?**

- Use the --set-upstream-to flag:

```
git branch --set-upstream-to=origin/branch-name
```

147. **What is the difference between a tracking branch and a local branch?**

- A **tracking branch** is a local branch linked to a remote branch, which makes it easier to pull and push changes.
- A **local branch** is any branch that exists only in your local repository.

148. **Scenario: How do you rename a branch locally and on the remote?**

- Rename locally:

```
git branch -m old-branch-name new-branch-name
```

- Rename remotely:
    1. Delete the old remote branch:

```
git push origin --delete old-branch-name
```

    2. Push the new branch:

```
git push origin new-branch-name
```

149. **Scenario: How do you cancel an incomplete merge?**

- Use the git merge --abort command:

```
git merge --abort
```

150. **What is the difference between a fast-forward merge and a three-way merge?**

- **Fast-forward merge:** Occurs when the branch being merged has not diverged, and Git can simply move the HEAD pointer forward.
- **Three-way merge:** Used when branches have diverged, requiring Git to create a new merge commit.

151. **Scenario: How do you perform a no-commit merge?**

- Use the --no-commit flag:

```
git merge --no-commit branch-name
```

152. **What are the risks of rebasing a public branch?**

- Rebasing rewrites history, which can lead to conflicts and issues for collaborators who are also working on the same branch. It should only be done on private branches.

153. **Scenario: How do you edit an old commit message during a rebase?**

- Steps:

1. Start an interactive rebase:

```
git rebase -i HEAD~n
```

2. Replace pick with reword for the desired commit.
3. Edit the message when prompted.

154. **How do you perform a rebase and automatically resolve conflicts in favor of one branch?**

- Use the --strategy-option=theirs flag:

```
git rebase -s recursive -X theirs branch-name
```

155. **Scenario: How do you revert changes in a specific commit while keeping later changes intact?**

- Use git revert:

```
git revert <commit-hash>
```

156. **What is a Git hook, and how can you implement one?**

- A Git hook is a script that runs automatically during specific Git events (e.g., pre-commit, post-merge).
- Example: Adding a pre-commit hook:

1. Create a file .git/hooks/pre-commit:

```
#!/bin/sh
echo "Running pre-commit hook"
```

2. Make it executable:

```
chmod +x .git/hooks/pre-commit
```

157. **Scenario: How do you handle a force-push on a shared branch?**

- Steps:

1. Fetch the latest changes:

```
git fetch origin
```

2. Reset your local branch:

```
git reset --hard origin/branch-name
```

158. **How do you fetch changes for a single file from a remote branch?**

- Use git checkout with the branch and file path:

```
git checkout origin/branch-name -- <file-path>
```

159. **Scenario: How do you create multiple stashes with custom messages?**

- Use the git stash save command with a message:

```
git stash save "Stash message 1"
git stash save "Stash message 2"
```

160. **How do you apply a stash without removing it from the stash list?**

- Use:

```
git stash apply stash@{n}
```

161. **Scenario: How do you split a repository into two smaller repositories?**

- Use git filter-repo:
    1. Install the tool:

```
pip install git-filter-repo
```

    2. Split the repository:

```
git filter-repo --path subfolder-name --force
```

162. **What is the purpose of the git worktree command?**

- The git worktree command allows you to work on multiple branches in the same repository without switching branches.
- Example:

```
git worktree add ../new-worktree branch-name
```

163. **Scenario: How do you fix "fatal: Authentication failed" when using HTTPS?**

- This error often occurs due to incorrect credentials or token expiration.
- Solutions:
    1. Update your credentials:

```
git credential-cache exit
```

Then re-enter your credentials on the next pull or push.

    2. Use a personal access token (PAT) instead of a password if the service requires it:
        - Generate a PAT from your Git hosting platform (e.g., GitHub, GitLab).
        - Use the PAT as the password when prompted.

164. **Scenario: How do you fix "index.lock" errors when performing Git operations?**

- This error occurs if a Git process was interrupted, leaving a lock file behind.
- Solution:
    1. Verify no Git processes are running:

```
ps aux | grep git
```

    2. Remove the lock file:

```
rm -f .git/index.lock
```

165. **Scenario: What do you do if a commit has the wrong author information?**

- Use git commit --amend to correct the author for the last commit:

```
git commit --amend --author="Name <email@example.com>"
```

- For multiple commits, use:

```
git rebase -i HEAD~n
```

Then edit each commit's author.

166. **Scenario: How do you troubleshoot "detected inconsistent line endings in file"?**

- Configure Git to handle line endings:
    1. Set core.autocrlf for your platform:
        - On Windows:

```
git config --global core.autocrlf true
```

        - On Linux/macOS:

```
git config --global core.autocrlf input
```

    2. Normalize the file's line endings:

```
git add --renormalize .
```

```
git commit -m "Normalize line endings"
```

167. **What does git reflog do, and how can it help recover lost commits?**

- git reflog tracks changes to HEAD, allowing you to recover commits that are no longer reachable via branches.
- Example:
    1. List the reflog:

```
git reflog
```

    2. Recover a lost commit:

```
git checkout <commit-hash>
```

168. **What is the purpose of git cherry-pick?**

- git cherry-pick applies a specific commit from one branch to another without merging the entire branch.
- Example:

```
git cherry-pick <commit-hash>
```

169. **How do you view all files in a specific commit?**

- Use:

```
git show --name-only <commit-hash>
```

170. **What is the difference between git log and git show?**

- **git log:** Shows a history of commits.
- **git show:** Displays detailed information about a specific commit, including changes made.

171. **Scenario: How do you merge changes from a specific branch into your branch without a full merge?**

- Use git cherry-pick to apply specific commits:

```
git cherry-pick <commit-hash>
```

172. **Scenario: How do you delete a remote branch?**

- Command:

```
git push origin --delete branch-name
```

173. **What is the difference between git branch and git rev-parse?**

- **git branch:** Lists, creates, or deletes branches.
- **git rev-parse:** Converts branch names or references into their SHA-1 hashes.

174. **Scenario: How do you avoid creating a merge commit for trivial changes?**

- Use a fast-forward merge by ensuring the branch has no divergent changes:

```
git merge --ff-only branch-name
```

175. **Scenario: How do you force a merge commit even when a fast-forward merge is possible?**

- Use the --no-ff flag:

```
git merge --no-ff branch-name
```

176. **Scenario: How do you merge only specific files from another branch?**

- Steps:
    1. Check out the files:

```
git checkout branch-name -- file-path
```

2. Stage and commit the changes:

```
git add file-path
```

```
git commit -m "Merged specific file from branch-name"
```

177. **Scenario: How do you interactively rebase to reorder commits?**

- Steps:

1. Start the interactive rebase:

```
git rebase -i HEAD~n
```

2. Change the order of the commits in the editor.
3. Save and exit.

178. **Scenario: What do you do if a rebase introduces a bug?**

- Abort the rebase and return to the pre-rebase state:

```
git rebase --abort
```

179. **What is the difference between rebasing and cherry-picking?**

- **Rebasing:** Reapplies a series of commits onto a new base, modifying commit history.
- **Cherry-picking:** Applies specific commits to another branch without altering history.

180. **Scenario: How do you work on a hotfix in Gitflow?**

- Steps:

1. Create a hotfix branch:

```
git checkout -b hotfix/fix-name main
```

2. Make changes and commit them.
3. Merge the hotfix into main and develop:

```
git checkout main
git merge hotfix/fix-name
git checkout develop
git merge hotfix/fix-name
```

181. **What are Git tags used for in workflows?**

- Tags mark specific points in history, often for releases.
- Example:

```
git tag -a v1.0 -m "Version 1.0"
```

```
git push origin v1.0
```

182. **Scenario: How do you resolve conflicts when merging a pull request?**

- Steps:

1. Pull the PR locally:

```
git fetch origin pull/<PR-number>/head:pr-branch
```

2. Switch to the branch and resolve conflicts manually.
3. Push the resolved branch:

```
git push origin pr-branch
```

183. **How do you create a pull request in GitHub using Git commands?**

- Push a branch:

```
git push origin branch-name
```

- Use the GitHub CLI to create a PR:

```
gh pr create --title "PR Title" --body "PR Description"
```

184. **Scenario: How do you stash changes for a specific file only?**

- Command:

```
git stash push <file-path>
```

185. **Scenario: How do you preview what a stash contains before applying it?**

- Command:

```
git stash show stash@{n} --patch
```

186. **Scenario: How do you manage submodules in a large repository?**

- Add a submodule:

```
git submodule add <repo-url> <path>
```

- Update submodules:

```
git submodule update --init --recursive
```

187. **What is the purpose of the git fsck command?**

- git fsck checks the integrity of a Git repository.
- Example:

```
git fsck
```

188. **Scenario: How do you handle the "detached HEAD" state after checking out a commit directly?**

- If you want to create a new branch from this state:

```
git checkout -b new-branch
```

- If you want to return to an existing branch:

```
git checkout branch-name
```

189. **Scenario: What do you do if you accidentally staged changes to the wrong file?**

- Unstage the file:

```
git reset <file-path>
```

- Make the necessary adjustments and restage the correct file(s):

```
git add correct-file-path
```

190. **Scenario: How do you resolve "fatal: origin does not appear to be a Git repository"?**

- This error occurs when the remote URL is missing or incorrect.
- Solution:
    1. Verify the remote:

```
git remote -v
```

    2. Add or correct the remote URL:

```
git remote add origin <repository-url>
```

191. **Scenario: How do you fix the "merge failed" error when rebasing?**

- Steps:
    1. Identify conflicting files:

```
git status
```

    2. Resolve conflicts manually.
    3. Stage the resolved files:

```
git add <file>
```

    4. Continue the rebase:

```
git rebase --continue
```

192. **How do you count the number of commits in a branch?**

- Command:

```
git rev-list --count branch-name
```

193. **What does the HEAD^ and HEAD~n syntax mean?**

- **HEAD^:** Refers to the immediate parent of the current commit.
- **HEAD~n:** Refers to the nth parent commit from the current commit.

194. **How do you display the current branch name?**

- Command:

```
git branch --show-current
```

195. **How do you display detailed information about a specific commit?**

- Command:

```
git show <commit-hash>
```

196. **Scenario: How do you ensure no one pushes directly to main?**

- Set up branch protection rules on your Git hosting platform (e.g., GitHub or GitLab).
- Alternatively, you can create a server-side pre-receive hook:

```
#!/bin/sh
if [ "$GIT_BRANCH" = "main" ]; then
  echo "Direct pushes to main are not allowed!"
  exit 1
fi
```

197. **Scenario: How do you archive a feature branch that is no longer in use?**

- Steps:
    1. Push the branch to the remote if not already done:

```
git push origin feature-branch
```

    2. Delete it locally:

```
git branch -d feature-branch
```

    3. Move it to an archive remote repository, if needed.

198. **Scenario: How do you identify and merge only the commits that add a specific file or feature?**

- Use git log to identify the commits:

```
git log -- <file-path>
```

- Cherry-pick those commits:

```
git cherry-pick <commit-hash>
```

199. **Scenario: How do you merge branches when there are too many conflicts?**

- Steps:
    1. Perform a merge:

```
git merge branch-name
```

    2. Use a merge tool to simplify conflict resolution:

```
git mergetool
```

3. Resolve the conflicts manually if necessary and commit:

```
git commit
```

200. **Scenario: How do you track changes made to Git submodules?**

- Commands:

1. Update the submodule:

```
git submodule update --remote
```

2. Commit the submodule changes:

```
git add <submodule-path>
git commit -m "Updated submodule"
```