

Git Interview Questions Guide

You said:



Give me the 50 most common interview questions for beginners, 50 for intermediate, and 25 for advanced.

A total of 125 questions.

Also, answer each question IN DETAIL in layman's terms.

Topic: Git

ChatGPT said:



Certainly! Let's dive into the 125 Git-related interview questions divided into beginner, intermediate, and advanced levels. I'll provide a detailed explanation in layman's terms for each.

Beginner-Level Git Questions (50 Questions)

1. What is Git?

- **Answer:** Git is a version control system that tracks changes in your files. It helps you collaborate with others and keeps a history of all changes so that you can revert to previous versions if needed.

2. What is a repository in Git?

- **Answer:** A repository (repo) is a storage location for your project files and their history. It can be local (on your computer) or remote (on platforms like GitHub).

3. How do you initialize a new Git repository?

- **Answer:** Use the command `git init` in your project folder. This creates a `.git` directory that starts tracking your files.

4. What is the difference between `git clone` and `git init`?

- **Answer:** `git init` creates a new Git repository in your existing project. `git clone` copies an existing remote repository to your local machine.

5. How do you check the status of your files in Git?

- **Answer:** Use the command `git status`. It shows which files have been modified, staged, or are untracked.

6. What does `git add` do?

- **Answer:** `git add` stages your changes, meaning it prepares your modified files to be committed. It tells Git, "I want to save these changes."

7. What is a commit in Git?

- **Answer:** A commit is a snapshot of your project at a certain point in time. It saves the current state of your files in the repository.

8. How do you commit changes in Git?

- **Answer:** Use the command `git commit -m "your message"`. The message describes what changes were made.

9. **What is the purpose of a commit message?**
 - **Answer:** The commit message explains what changes were made in the commit, helping you and others understand the history of the project.
10. **How do you view commit history?**
 - **Answer:** Use the command ``git log``. It shows a list of all commits with their messages, authors, and timestamps.
11. **What is the difference between ``git pull`` and ``git fetch``?**
 - **Answer:** ``git fetch`` downloads changes from the remote repository but doesn't merge them. ``git pull`` does both fetching and merging in one step.
12. **What is a branch in Git?**
 - **Answer:** A branch is a separate line of development. You can create branches to work on new features without affecting the main project.
13. **How do you create a new branch?**
 - **Answer:** Use the command ``git branch branch_name``. Replace ``branch_name`` with your desired branch name.
14. **How do you switch to a different branch?**
 - **Answer:** Use ``git checkout branch_name``. This moves you to the specified branch.
15. **What is merging in Git?**
 - **Answer:** Merging combines changes from different branches into one. It's how you bring features from separate branches together.
16. **How do you merge branches?**
 - **Answer:** First, switch to the branch you want to merge into using ``git checkout branch_name``, then use ``git merge other_branch_name``.
17. **What is a conflict in Git?**
 - **Answer:** A conflict happens when two branches have changes in the same place, and Git doesn't know which one to keep. You have to resolve it manually.
18. **How do you resolve a merge conflict?**
 - **Answer:** Open the conflicting file, decide which changes to keep, edit the file, then stage and commit the resolved file.
19. **What is ``.gitignore``?**
 - **Answer:** A ``.gitignore`` file tells Git which files or folders to ignore and not track. This is useful for things like temporary files or sensitive information.
20. **How do you create a ``.gitignore`` file?**
 - **Answer:** Simply create a new file named ``.gitignore`` in your project's root directory and list the files or folders you want Git to ignore.
21. **What is a remote repository?**
 - **Answer:** A remote repository is a version of your project stored on a server (like GitHub or GitLab) that you can share with others.
22. **How do you add a remote repository?**
 - **Answer:** Use ``git remote add origin url``, replacing ``url`` with the link to your remote repository.
23. **What does ``git push`` do?**
 - **Answer:** ``git push`` uploads your local changes (commits) to the remote repository, making them available to others.
24. **How do you rename a branch?**
 - **Answer:** Use ``git branch -m old_name new_name``. This renames the branch.
25. **How do you delete a branch?**
 - **Answer:** Use ``git branch -d branch_name`` for a safe delete (only if the branch is fully merged), or ``git branch -D branch_name`` to force delete.

26. What is the `HEAD` in Git?

- **Answer:** `HEAD` is a reference to the last commit in the current branch. It tells you where you are in the commit history.

27. How do you undo a commit?

- **Answer:** You can use `git revert commit_id` to create a new commit that undoes the changes of a previous commit. Alternatively, use `git reset` for more powerful undo options.

28. What does `git stash` do?

- **Answer:** `git stash` temporarily saves your uncommitted changes so you can work on something else, then apply them back later.

29. How do you apply stashed changes?

- **Answer:** Use `git stash apply` to reapply the most recent stashed changes, or `git stash pop` to apply and delete the stash.

30. How do you delete a stash?

- **Answer:** Use `git stash drop` to remove a specific stash or `git stash clear` to delete all stashes.

31. What does `git diff` do?

- **Answer:** `git diff` shows the differences between your files and the last commit or between different commits, helping you see what's changed.

32. How do you remove a file from Git without deleting it from your file system?

- **Answer:** Use `git rm --cached filename`. This removes the file from tracking but keeps it on your computer.

33. What is `git blame`?

- **Answer:** `git blame` shows who made changes to each line in a file, helping you trace back the history of specific code.

34. What is `git cherry-pick`?

- **Answer:** `git cherry-pick` applies changes from a specific commit to your current branch, even if the commit is from another branch.

35. How do you rename a file in Git?

- **Answer:** Use `git mv old_filename new_filename`. This renames the file and stages the change for commit.

36. What is a tag in Git?

- **Answer:** A tag is a reference to a specific commit, often used to mark releases or important points in the project.

37. How do you create a tag?

- **Answer:** Use `git tag tag_name` to create a tag, or `git tag -a tag_name -m "message"` to create an annotated tag with a message.

38. How do you delete a tag?

- **Answer:** Use `git tag -d tag_name` to delete a local tag. To delete a remote tag, use `git push origin :refs/tags/tag_name`.

39. How do you check out a specific commit?

- **Answer:** Use `git checkout commit_id`. This moves you to the state of the project at that specific commit.

40. What is `git rebase`?

- **Answer:** `git rebase` re-applies commits from one branch onto another, creating a linear history. It's useful for keeping your commit history clean.

41. What is a detached HEAD?

- **Answer:** A detached HEAD occurs when you're not on a branch but on a specific commit. Any changes you make won't belong to a branch unless you create one.

42. How do you move a commit to another branch?

- **Answer:** You can use ``git cherry-pick commit_id`` to apply a commit from one branch to another.

43. What is ``git log --oneline``?

- **Answer:** This command shows a condensed version of your commit history with one line per commit.

44. What does ``git clean`` do?

- **Answer:** ``git clean`` removes untracked files from your working directory. Be careful, as this cannot be undone.

45. What is ``git reflog``?

- **Answer:** ``git reflog`` records every change to the HEAD, allowing you to recover from mistakes, like deleting a branch.

46. What does ``git config`` do?

- **Answer:** ``git config`` sets configuration options, such as your name and email, or aliasing commands for easier use.

47. How do you set your username and email in Git?

- **Answer:** Use ``git config --global user.name "Your Name"`` and ``git config --global user.email "your_email@example.com"``.

48. What is a fast-forward merge?

- **Answer:** A fast-forward merge happens when there's no divergence in the commit history, so Git can simply move the branch pointer forward.

49. What does ``git reset`` do?

- **Answer:** ``git reset`` undoes changes by moving the branch pointer to a different commit. It can modify your working directory, index, or both.

50. What is the difference between ``git reset --soft`` and ``git reset --hard``?

- **Answer:** ``git reset --soft`` keeps your changes but moves the commit pointer, while ``git reset --hard`` discards all changes and resets everything.

Intermediate-Level Git Questions (50 Questions)

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

- **Answer:** ``git merge`` combines branches, creating a merge commit, while ``git rebase`` moves your commits on top of another branch, resulting in a linear history.

2. What is a Git hook?

- **Answer:** A Git hook is a script that runs automatically when certain Git events occur, like committing or pushing. Hooks can enforce policies or automate tasks.

3. How do you create and use a Git hook?

- **Answer:** In the ``.git/hooks`` directory, create a file like ``pre-commit`` and add your script. Make it executable with ``chmod +x pre-commit``.

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

- **Answer:** ``git stash apply`` re-applies the stashed changes without removing them from the stash, while ``git stash pop`` applies and deletes them.

5. What is the difference between ``git reset`` and ``git revert``?

- **Answer:** ``git reset`` moves the branch pointer and can undo changes, while ``git revert`` creates a new commit that undoes the effects of a previous commit without altering history.

6. What is a submodule in Git?

- **Answer:** A submodule is a repository within another repository. It allows you to keep a separate project inside your main project.

7. How do you add a submodule to your Git repository?

- **Answer:** Use ``git submodule add url path``, where ``url`` is the submodule repository, and ``path`` is where you want it in your project.

8. How do you update a submodule?

- **Answer:** Use ``git submodule update --remote`` to pull the latest changes from the submodule's repository.

9. How do you remove a submodule?

- **Answer:** First, remove the submodule entry in ``.gitmodules``, then use ``git rm --cached path`` to remove it from Git and delete the submodule directory.

10. What is ``git bisect``?

- **Answer:** ``git bisect`` is a tool to find the commit that introduced a bug by performing a binary search through the commit history.

11. How do you use ``git bisect``?

- **Answer:** Start with ``git bisect start``, then mark a good commit with ``git bisect good`` and a bad one with ``git bisect bad``. Git will check out different commits until the bad commit is found.

12. What is ``git cherry``?

- **Answer:** ``git cherry`` helps find commits that are in one branch but not in another, showing what's different between branches.

13. What is ``git filter-branch``?

- **Answer:** ``git filter-branch`` rewrites the commit history of your repository, allowing you to remove sensitive data, change commit messages, or modify files across all commits.

14. How do you track a remote branch?

- **Answer:** Use ``git branch --track branch_name origin/branch_name`` or ``git checkout -b branch_name origin/branch_name`` to set up tracking.

15. What does ``git remote -v`` do?

- **Answer:** This command shows the URLs of the remote repositories you've connected to your local repository.

16. How do you remove a remote repository?

- **Answer:** Use ``git remote remove remote_name``. This disconnects your local repository from the remote one.

17. How do you rename a remote repository?

- **Answer:** Use ``git remote rename old_name new_name``. This changes the reference name of the remote repository.

18. What is a detached HEAD state in Git, and how do you recover from it?

- **Answer:** A detached HEAD occurs when you check out a specific commit instead of a branch. Recover by switching back to a branch with ``git checkout branch_name``.

19. What is ``git squash``?

- **Answer:** ``git squash`` refers to combining multiple commits into a single commit to make your commit history cleaner.

20. How do you squash commits in Git?

- **Answer:** During an interactive rebase with ``git rebase -i``, change ``pick`` to ``squash`` for the commits you want to combine.

21. How do you change the message of an older commit?

- **Answer:** Use ``git rebase -i`` to enter interactive rebase mode, then choose ``reword`` for the commit you want to edit.

22. What is ``git shortlog``?

- **Answer:** ``git shortlog`` groups and summarizes the commit history by author, making it easier to see contributions.

23. How do you undo a ``git push``?

- **Answer:** You can use ``git revert`` to undo the changes, then push the new commit. Alternatively, you can force push (``git push -f``) after resetting to the previous state, but this rewrites history and should be done with caution.

24. What does ``git pull --rebase`` do?

- **Answer:** This command fetches changes from the remote branch and then rebases your local commits on top of the remote changes, avoiding merge commits.

25. What is a ``bare`` repository in Git?

- **Answer:** A bare repository is a repository without a working directory. It's used for sharing changes and doesn't allow direct modification of files.

26. How do you create a bare repository?

- **Answer:** Use ``git init --bare``. This creates a repository intended for sharing, without a working directory.

27. How do you undo the last commit but keep the changes?

- **Answer:** Use ``git reset --soft HEAD~1``. This removes the last commit but leaves your changes in the staging area.

28. What is the difference between ``git fetch`` and ``git pull --rebase``?

- **Answer:** ``git fetch`` downloads changes without merging them. ``git pull --rebase`` fetches and then rebases your changes on top of the fetched changes.

29. What is ``git prune``?

- **Answer:** ``git prune`` removes objects that are no longer referenced in your repository, like dangling commits or unreachable branches.

30. What does ``git gc`` do?

- **Answer:** ``git gc`` runs a garbage collection to clean up unnecessary files and optimize the repository.

31. What is ``git fsck``?

- **Answer:** ``git fsck`` checks the integrity of your repository and identifies issues like corrupted objects or unreachable commits.

32. How do you track the origin of a commit?

- **Answer:** Use ``git log --follow filename`` to track changes to a specific file or ``git blame`` to see who made changes to specific lines.

33. What is a Git bundle?

- **Answer:** A Git bundle is a package of repository data that can be transferred and applied to another repository, useful for offline work.

34. How do you create a Git bundle?

- **Answer:** Use ``git bundle create filename branch_name``. This creates a bundle file containing the specified branch.

35. What is a reflog in Git?

- **Answer:** Reflog (reference log) tracks every movement of your ``HEAD``, allowing you to recover from mistakes, like resetting the wrong branch.

36. How do you recover a deleted branch in Git?

- **Answer:** Use ``git reflog`` to find the commit where the branch was last present, then use ``git checkout -b branch_name commit_id`` to recreate the branch.

37. What does ``git archive`` do?

- **Answer:** ``git archive`` creates a compressed archive (e.g., zip or tar) of your repository or a specific branch, useful for sharing or backups.

38. What is ``git am``?

- **Answer:** ``git am`` applies patches from an email or file, allowing you to import commits created by others.

39. What is ``git format-patch``?

- **Answer:** ``git format-patch`` creates patch files for commits, useful for sending changes via email or sharing with others.
- 40. **What is ``git apply``?**
 - **Answer:** ``git apply`` applies changes from a patch file without committing them, allowing you to test changes before making them permanent.
- 41. **How do you cancel a rebase?**
 - **Answer:** Use ``git rebase --abort`` to cancel the rebase and return to the state before the rebase started.
- 42. **What is ``git worktree``?**
 - **Answer:** ``git worktree`` allows you to work on multiple branches simultaneously by creating separate working directories linked to the same repository.
- 43. **How do you create a new worktree?**
 - **Answer:** Use ``git worktree add path branch_name`` to create a new working directory for the specified branch.
- 44. **What is ``git rerere``?**
 - **Answer:** ``git rerere`` stands for "reuse recorded resolution." It remembers how you resolved conflicts, so if the same conflict occurs again, Git can automatically resolve it.
- 45. **How do you enable ``git rerere``?**
 - **Answer:** Use ``git config --global rerere.enabled true``. This turns on the feature for all repositories.
- 46. **How do you split a commit in Git?**
 - **Answer:** Use ``git reset HEAD^`` to undo the commit, then use ``git add -p`` to selectively stage changes and commit them in smaller parts.
- 47. **How do you combine two repositories into one?**
 - **Answer:** Add one repository as a remote in the other using ``git remote add``, then fetch and merge the changes from that remote.
- 48. **What is the ``ORIG_HEAD`` in Git?**
 - **Answer:** ``ORIG_HEAD`` is a reference to the previous state of ``HEAD`` before the last operation that changed it, such as a merge or reset.
- 49. **What does ``git push --force-with-lease`` do?**
 - **Answer:** This command force-pushes your changes but only if no one else has pushed new changes since you last pulled, preventing accidental overwriting.
- 50. **How do you amend a commit without changing its message?**
 - **Answer:** Use ``git commit --amend --no-edit``. This modifies the last commit without altering the commit message.

Advanced-Level Git Questions (25 Questions)

1. **What is ``git subtree``?**
 - **Answer:** ``git subtree`` allows you to split or merge repositories by treating a subdirectory as a separate repository, useful for managing dependencies.
2. **How do you use ``git subtree``?**
 - **Answer:** To add a subtree, use ``git subtree add --prefix=subdir url branch``. To split it out, use ``git subtree split --prefix=subdir``.
3. **What is ``git rev-parse``?**
 - **Answer:** ``git rev-parse`` converts references (like branch names or commit IDs) into their SHA-1 hashes, useful for scripting and automation.
4. **What is the difference between ``git checkout -b`` and ``git switch -c``?**

- **Answer:** Both commands create and switch to a new branch, but `git switch` is a more recent, simplified command designed to replace some of `git checkout`'s functions.

5. What is `git fsck` and when should you use it?

- **Answer:** `git fsck` checks the integrity of your repository and identifies issues like corrupted objects or unreachable commits. Use it to ensure your repository is healthy.

6. How do you handle large binary files in Git?

- **Answer:** Use Git LFS (Large File Storage) to track large files without slowing down your repository. It stores large files outside of your repository while keeping references in Git.

7. How do you optimize a Git repository?

- **Answer:** Use `git gc` for garbage collection and `git repack` to reduce the size of your repository. These commands clean up and compress unnecessary files.

8. How do you handle repository corruption in Git?

- **Answer:** First, use `git fsck` to identify corrupted objects. You can try recovering them from backups or using `git reflog` to restore your repository to a healthy state.

9. What is `git rebase -i` and how do you use it?

- **Answer:** `git rebase -i` (interactive rebase) allows you to modify commit history by editing, squashing, or reordering commits. Use it to clean up your commit history before merging.

10. What is `git rerere` and how does it work?

- **Answer:** `git rerere` records how you resolve merge conflicts, so Git can automatically apply the same resolution if the conflict happens again.

11. How do you rewrite history in Git?

- **Answer:** Use `git rebase -i` or `git filter-branch` to change past commits. Be careful with rewriting history, especially in shared repositories, as it can cause conflicts for others.

12. How do you handle a Git rebase gone wrong?

- **Answer:** Use `git rebase --abort` to cancel the rebase and return to the state before it started. If the rebase is partially completed, you can use `git reflog` to recover.

13. What is `git reflog` and how can it save you?

- **Answer:** `git reflog` records every move of your `HEAD`, allowing you to recover lost commits, branches, or changes, even after a reset or rebase.

14. How do you automate Git tasks using hooks?

- **Answer:** Git hooks are scripts triggered by Git events like commits or pushes. You can automate tasks like running tests, formatting code, or sending notifications by placing scripts in the `.git/hooks` directory.

15. What is the difference between `git fetch` and `git pull` in terms of workflow?

- **Answer:** `git fetch` only updates your local references without changing your working directory, allowing you to review changes before merging. `git pull` fetches and merges in one step, which can cause unexpected conflicts.

16. How do you create a custom Git command?

- **Answer:** Create a script in your `PATH` with the name `git-<command>`. Git will recognize and execute it as a custom command. For example, a script named `git-hello` can be run with `git hello`.

17. What is `git rev-list` and how is it used?

- **Answer:** `git rev-list` lists commit objects in reverse chronological order. It's used for scripting and filtering commits, like finding all commits between two tags.

18. How do you use `git notes`?

- **Answer:** `git notes` allows you to attach additional information to commits without changing the commit itself. Use `git notes add` to add a note and `git notes show` to view it.

19. What is the difference between ``git gc --aggressive`` and ``git gc``?

- **Answer:** ``git gc --aggressive`` performs a more thorough garbage collection, compressing objects more than the regular ``git gc``. It can reduce repository size but takes longer.

20. How do you work with multiple remotes in Git?

- **Answer:** Add multiple remotes using ``git remote add remote_name url``. You can fetch from or push to different remotes by specifying the remote name in your commands.

21. What is ``git blame -L``?

- **Answer:** ``git blame -L`` limits the output of ``git blame`` to specific lines of a file, useful for investigating changes in a particular section of code.

22. What is ``git rev-parse --show-toplevel``?

- **Answer:** This command shows the absolute path of the top-level directory of your Git repository, helping you find the root of your project.

23. How do you reapply changes from a commit in a different branch?

- **Answer:** Use ``git cherry-pick commit_id`` to apply the changes from a specific commit to your current branch, even if the commit is from another branch.

24. What does ``git submodule update --init`` do?

- **Answer:** This command initializes and updates submodules in your repository, ensuring they are checked out and up to date.

25. How do you track changes to Git configuration?

- **Answer:** Git configuration changes are stored in ``.git/config``, `~/ .gitconfig``, or ``/etc/gitconfig``. Use ``git config --list`` to see the current configuration, and version-control your global config files if necessary.

These questions cover a range of Git topics, from basic commands to advanced features, ensuring a comprehensive understanding of Git for interviews at all levels. The detailed explanations should make each concept easier to grasp, even for those who are new to Git.

ChatGPT can make mistakes. Check important info.