Git Guide: 50 Common Questions and Advanced Concepts

July 6, 2025

Contents

1 Introduction

This guide provides answers to 50 commonly asked questions about Git, covering beginner, intermediate, and advanced topics. It is designed for users of all levels to understand Git's core concepts, workflows, and advanced techniques. Each question is answered concisely, with practical examples where applicable.

2 Beginner Questions (1–20)

Q1. What is Git?: Git is a distributed version control system that tracks changes in source code, allowing multiple developers to collaborate efficiently. **Q2. How do I install Git?:** Download Git from `https://git-scm.com` and follow the installation instructions for your operating system (Windows, macOS, or Linux). **Q3. How do I initialize a Git repository?:** Run `git init` in your project directory to create a new Git repository. **Q4. What is a Git repository?:** A repository is a storage space where Git tracks your projects files and their history. **Q5. How do I check the status of my repository?:** Use `git status` to see the current state of your working directory and staging area. **Q6. What is the staging area?:** The staging area (or index) is an intermediate area where changes are prepared before committing. **Q7. How do I stage changes?:** Use `git add <file>` to stage specific files or `git add .` to stage all changes. **Q8. How do I commit changes?:** Run `git commit -m "commit message"` to save staged changes with a descriptive message. **Q9. What is a commit?:** A commit is a snapshot of your projects changes at a specific point in time. **Q10. How do I view commit history?:** Use `git log` to see a list of commits, or `git log -oneline` for a condensed view. **Q11. How do I create a branch?:** Run `git branch <branch-name>` to create a new branch. **Q12. How do I switch branches?:** Use `git checkout <branch-name>` or `git switch <branch-name>` (Git 2.23+). **Q13. How do I merge branches?:** Run `git merge <branch-name>` while on the target branch to combine changes. **Q14. What is a merge conflict?:** A merge conflict occurs when Git cannot automatically reconcile changes from two branches. **Q15. How do I resolve a merge conflict?:** Manually edit the conflicting files, mark them as resolved with `git add`, and then commit. **Q16. How do I push changes to a remote repository?:** Use `git push origin <branch-name>` to send local commits to a remote repository. **Q17. How do I pull changes from a remote repository?:** Run `git pull origin <branch-name>` to fetch and merge remote changes. **Q18. What is a remote repository?:** A remote repository is a version of your project hosted on a server, like GitHub or GitLab. **Q19. How do I clone a repository?:** Use `git clone <repository-url>` to copy a remote repository to your local machine. **Q20. How do I ignore files in Git?:** Create a `.gitignore` file and list files or patterns to exclude from tracking.

3 Intermediate Questions (21–40)

Q21. What is the difference between `git fetch` and `git pull`?: `git fetch` downloads remote changes without merging, while `git pull` fetches and merges. **Q22. How do I undo a commit?:** Use `git revert <commit-hash>` to create a new commit undoing the specified commit, or `git reset -soft <commit-hash>` to undo without losing changes. **Q23. What is `git stash`?:** `git stash` temporarily saves uncommitted changes, allowing you to switch branches. **Q24. How do I apply a stash?:** Run `git stash apply` to restore the most recent stash, or `git stash apply stash@{n}` for a specific stash. **Q25. How do I delete a branch?:** Use `git branch -d <branch-name>` for local branches, or `git push origin -delete <branch-name>` for remote branches. **Q26. What is a pull request?:** A pull request is a request to merge changes from one branch to another, often used in platforms like GitHub. **Q27. How do I rebase a branch?:** Run `git rebase <base-branch>` to move your branchs commits onto the tip of the base branch. **Q28. What is the difference between merge and rebase?:** Merge combines branches, preserving history, while rebase rewrites history for a linear timeline. **Q29. How do I amend a commit?:** Use `git commit -amend` to modify the most recent commits message or contents. **Q30. How do I view changes in a file?:** Run `git diff <file>` to see uncommitted changes, or `git diff <commit1> <commit2> <file>` for specific commits. **Q31. How do I configure my Git username?:** Set it globally with `git config -global user.name "Your Name"`. **Q32. How do I set up a remote repository?:** Add a remote with `git remote add origin <repository-url>`. **Q33. How do I check remote details?:** Use `git remote -v` to list remote repositories and their URLs. **Q34. What is `git cherry-pick`?:** `git cherry-pick <commit-hash>` applies a specific commit to the current branch. **Q35. How do I squash commits?:** Use `git rebase -i <commit-hash>` and mark commits to squash, then edit the commit message. **Q36. How do I reset to a previous commit?:** Run `git reset -hard <commit-hash>` to discard changes, or `git reset -soft` to keep them. **Q37. How do I tag a**

commit?: Use `git tag <tag-name> <commit-hash>` to create a tag, and `git push origin <tag-name>` to share it. **Q38. What is a detached HEAD state?:** A detached HEAD occurs when you check out a commit or tag instead of a branch. **Q39. How do I recover a deleted branch?:** Use `git reflog` to find the branchs last commit hash, then `git branch <branch-name> <commit-hash>`. **Q40. How do I set up SSH keys for Git?:** Generate a key with `ssh-keygen`, add it to your SSH agent, and upload the public key to your Git hosting service.

4 Advanced Questions and Concepts (41–50)

Q41. What is `git bisect` and how is it used?: `git bisect` helps find the commit that introduced a bug by binary searching through commit history. Start with `git bisect start`, mark a good and bad commit, and Git narrows it down. **Q42. How do I use `git worktree`?:** `git worktree` allows multiple working directories for a single repository. Create one with `git worktree add <path> <branch>`. **Q43. What is a bare repository?:** A bare repository contains only the `.git` directory, used for remote repositories without a working tree. **Q44. How do I optimize a large repository?:** Run `git gc` to clean up unused objects and compress the repository, or use `git lfs` for large files. **Q45. What is `git reflog` and how is it used?:** `git reflog` tracks reference updates, useful for recovering lost commits or branches. View it with `git reflog`. **Q46. How do I rewrite history with `git filter-branch`?:** `git filter-branch` modifies commit history, e.g., to remove sensitive data. Example: `git filter-branch -tree-filter ’rm -f <file>’ HEAD`. **Q47. What are Git hooks?:** Git hooks are scripts that run automatically on specific events, like pre-commit or post-merge, stored in `.git/hooks`. **Q48. How do I set up a Git hook?:** Create a script in `.git/hooks`, e.g., `pre-commit`, and make it executable with `chmod +x pre-commit`. **Q49. What is `git submodule` and how is it used?:** `git submodule` manages nested repositories. Add one with `git submodule add <repository-url>`, and update with `git submodule update`. **Q50. How do I configure Git for a monorepo?:** Use sparse-checkout (`git sparse-checkout init`) and partial clones (`git clone -filter=blob:none`) to manage large monorepos efficiently.

5 Advanced Git Concepts

5.1 Git Rebase vs. Merge Workflows

Rebase creates a linear history but rewrites commits, while merge preserves history with merge commits. Use rebase for clean local branches and merge for shared branches to avoid conflicts.

5.2 Git LFS (Large File Storage)

Git LFS handles large files by storing them outside the main repository. Install Git LFS, then track files with `git lfs track "*.bin"` and commit the `.gitattributes` file.

5.3 Monorepo Management

Monorepos store multiple projects in one repository. Use `git sparse-checkout` to work on specific directories and `git worktree` for parallel development.

5.4 Git Hooks for Automation

Hooks automate tasks like linting (pre-commit) or deploying (post-merge). Example: a `pre-commit` hook to run tests:

```
#!/bin/sh
npm test
```

5.5 Git Bisect for Debugging

Use `git bisect` to automate bug hunting. Example: `git bisect start <bad-commit> <good-commit>`, then mark commits as `git bisect good` or `bad`.