GIT CHEAT SHEET & INTERVIEW GUIDE

Generated on: 2025-08-13 09:44:21

This document contains essential Git commands, explanations, and common interview questions with detailed answers.

Table of Contents

1. Git Commands by Category

Basic Commands

git init

Initialize a new Git repository in the current directory

git clone <repository>

Clone an existing repository from a remote server

git status

Show the working tree status (staged, unstaged, untracked files)

git add <file>

Add file contents to the staging area (index)

git add .

Add all changed files to the staging area

git commit -m "message"

Record changes to the repository with a descriptive message

git commit -a -m "message"

Add and commit all changed tracked files in one step

git log

Show commit logs

git diff

Show changes between working directory and staging area

git diff --staged

Show changes between staging area and last commit

Branching & Merging

git branch

List all local branches (current branch marked with *)

git branch <branch-name>

Create a new branch

git branch -d <branch-name>

Delete a branch (safe delete)

git branch -D <branch-name>

Force delete a branch (even if unmerged)

git checkout <branch-name>

Switch to another branch

git checkout -b <branch-name>

Create and switch to a new branch

git merge <branch-name>

Merge specified branch into current branch

git merge --abort

Abort a merge in progress

git rebase <branch-name>

Reapply commits on top of another branch

git rebase --abort

Abort a rebase in progress

Remote Repositories

git remote -v

List all remote repositories with URLs

git remote add <name> <url>

Add a new remote repository

git remote remove <name>

Remove a remote repository

git fetch <remote>

Download objects and refs from remote repository

git pull <remote> <branch>

Fetch and merge changes from remote branch

git push <remote> <branch>

Push local commits to remote branch

git push -u <remote> <branch>

Push and set upstream tracking reference

git push --force

Force push (use with caution!)

Undoing Changes

git restore <file>

Discard changes in working directory (unstaged changes)

git restore --staged <file>

Unstage a file (keep changes in working directory)

git reset --hard

Reset working directory and staging area to last commit (dangerous)

git reset --soft HEAD~1

Undo last commit but keep changes in staging area

git reset --mixed HEAD~1

Undo last commit and unstage changes (default)

git revert <commit>

Create a new commit that undoes changes from a specific commit

git commit --amend

Modify the last commit (change message or add forgotten files)

Stashing

git stash

Stash changes in working directory (save for later)

git stash list

List all stashed changesets

git stash pop

Apply and remove the most recently stashed changes

git stash apply

Apply stashed changes but keep them in stash list

git stash drop

Remove the most recently stashed changeset

git stash clear

Remove all stashed changesets

Tagging

git tag

List all tags

git tag <tag-name>

Create a lightweight tag at current commit

git tag -a <tag-name> -m "message"

Create an annotated tag with message

git tag -d <tag-name>

Delete a tag

git push <remote> --tags

Push all tags to remote repository

Advanced Commands

git bisect

Binary search through commits to find a bug

git cherry-pick <commit>

Apply changes from a specific commit

git reflog

Show reference logs (helps recover lost commits)

git gc

Cleanup unnecessary files and optimize local repository

git fsck

Verify integrity of Git file system

git submodule

Manage nested repositories within a repository

2. Interview Questions with Answers

1. What is Git and how does it differ from other version control systems?

Git is a distributed version control system that tracks changes in source code during software development. Unlike centralized VCS (like SVN), Git gives every developer a full copy of the repository history, enabling offline work and faster operations. Key differences include:
- Distributed nature (no single point of failure)
- Branching and merging are lightweight and fast
- Integrity through SHA-1 hashing
- Staging area for selective commits

2. Explain the Git workflow and the three main states of a file.

Git has three main states for files:
1. Modified: File is changed but not committed to database
2. Staged: File is marked to go into next commit snapshot (added to index)
3. Committed: File is safely stored in local database

The basic workflow:
1. Modify files in working directory
2. Stage changes (git add)
3. Commit changes (git commit) which takes staged snapshot permanently

3. What is the difference between 'git pull' and 'git fetch'?

git fetch only downloads new data from remote repository but doesn't integrate any of it into your working files. git pull is essentially git fetch followed by git merge - it fetches and immediately tries to merge the remote content with your local work.

4. How do you resolve a merge conflict in Git?

Steps to resolve merge conflicts:
1. Identify conflicted files (git status)
2. Open files and look for conflict markers (<<<<<<<, =======, >>>>>>>)
3. Edit files to keep desired changes (remove markers)
4. Add resolved files (git add)
5. Commit the resolution (git commit)

Tools like git mergetool can help visualize and resolve conflicts.

5. What is a 'detached HEAD' state and how do you recover from it?

A detached HEAD occurs when you check out a specific commit rather than a branch. Your HEAD points directly to a commit, not a branch reference. To recover:
1. Create a new branch: git checkout -b new-branch-name
2. Or switch back to a branch: git checkout main

Work done in detached HEAD will be lost unless you create a branch pointing to it.

6. Explain the difference between 'git rebase' and 'git merge'.

git merge creates a new commit that combines changes from both branches, preserving history as it happened.

git rebase rewrites history by moving the entire feature branch to begin at the tip of the main branch, creating linear history.

Key differences:
- Merge preserves history, rebase rewrites it
- Merge results in a merge commit, rebase doesn't
- Rebase creates cleaner, linear history
- Rebase can cause problems if shared branches are rebased

7. What is the '.gitignore' file and how is it used?

.gitignore is a text file that tells Git which files or folders to ignore in a project. It's used to prevent tracking of:
- Temporary files
- Build artifacts
- Local configuration files
- Sensitive data (credentials)

Patterns can include:
- Specific filenames (file.txt)
- Wildcards (*.log)
- Directories (build/)
- Negation (!important.log)

The file should be committed to the repository to share ignore rules with all team members.

8. How do you squash multiple commits into one?

To squash commits:
1. Use interactive rebase: git rebase -i HEAD~n (where n is number of commits)
2. In the editor, change 'pick' to 'squash' for commits you want to combine
3. Save and close the editor
4. Edit the new combined commit message

Alternatively, for recent commits: git reset --soft HEAD~n && git commit -m "new message"

9. What is a 'fast-forward' merge in Git?

A fast-forward merge occurs when there is a linear path from the current branch tip to the target branch. Instead of creating a merge commit, Git simply moves the branch pointer forward. This happens when:
- No divergent changes exist on the current branch
- The target branch is directly ahead of the current branch

You can prevent fast-forward with: git merge --no-ff

10. How do you completely remove a file from Git history?

To completely remove a file from history (including all commits):
1. Use git filter-branch or BFG Repo-Cleaner
2. Example: git filter-branch --force --index-filter 'git rm --cached --ignore-unmatch filename' --prune-empty --tag-name-filter cat -- --all
3. Force push to remote: git push origin --force --all

Warning: This rewrites history and affects all team members. Only do this for sensitive data.

11. What are Git hooks and how can they be used?

Git hooks are scripts that run automatically before or after specific Git commands. They are stored in .git/hooks directory. Common uses:
- pre-commit: Run tests/linters before commit
- pre-push: Run tests before pushing
- post-receive: Deploy code after push to server
- prepare-commit-msg: Modify commit messages

Hooks can be written in any scripting language. They are local to each repository and not version controlled by default (but can be shared via symlinks or a hooks directory in the project).

12. Explain the difference between 'git reset' and 'git revert'.

git reset moves the branch pointer backward, effectively erasing commits (can be dangerous if shared). It has three modes:
- --soft: Keeps changes in staging area
- --mixed (default): Keeps changes in working directory
- --hard: Discards all changes

git revert creates a new commit that undoes changes from a previous commit. This is safer for shared history as it doesn't rewrite existing commits.

13. How do you find a commit that introduced a specific bug?

Use git bisect for binary search through history:
1. Start bisect: git bisect start
2. Mark bad commit (current): git bisect bad
3. Mark good commit (known working): git bisect good <commit>
4. Git checks out middle commit - test it
5. Mark as good/bad: git bisect good/bad
6. Repeat until bug is found
7. Reset when done: git bisect reset

Alternative: git log -S"specific_code" to search commit contents

14. What is the purpose of 'git stash' and when would you use it?

git stash temporarily shelves changes so you can work on something else. Common use cases:
- Switching branches with uncommitted changes
- Pulling latest changes with uncommitted work
- Temporarily saving incomplete work

Commands:
- git stash: Save changes
- git stash list: View stashes
- git stash pop: Apply and remove most recent stash
- git stash apply: Apply without removing

Stashes are stack-based and can include untracked files with -u option.

15. How do you set up Git to ignore changes in file permissions?

To ignore file permission changes:
1. Set core.fileMode to false: git config core.fileMode false
2. For existing repositories, this can be set globally: git config --global core.fileMode false

This tells Git to ignore executable bit changes on files. Note: On Windows this is often automatically set to false as filesystem doesn't support Unix permissions.

16. What is the difference between 'HEAD', 'working tree', and 'index'?

- HEAD: Pointer to the last commit in the currently checked-out branch
- Working tree (working directory): The actual files you see and edit in your project directory
- Index (staging area): Intermediate area where you prepare commits

When you modify files, changes are in working tree. git add stages changes to index. git commit takes snapshot from index and moves HEAD to new commit.

17. How do you rename a Git branch locally and remotely?

To rename a branch:
1. Rename local branch: git branch -m old-name new-name
2. Push new branch to remote: git push origin new-name
3. Delete old remote branch: git push origin :old-name

If branch is checked out, use: git branch -m new-name

To update tracking: git push origin -u new-name

18. What is a 'bare repository' in Git?

A bare repository is a Git repository without a working directory. It contains only the version control information (in .git directory) and no checked-out files. Bare repos are used as:
- Central shared repositories (on servers)

- For cloning and pushing

Create with: git init --bare

They don't have working files, so you can't edit or commit directly in them.

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

To recover a deleted branch:

1. Find the commit SHA using git reflog
2. Create new branch at that commit: git branch branch-name <sha>

If not in reflog, you can:

1. Find the commit in merge/pull request history
2. Use git fsck to find dangling commits
3. Check remote if it was pushed: git checkout -b branch-name origin/branch-name

20. What are Git submodules and when would you use them?

Submodules allow embedding one Git repository inside another. Use cases:

- Including libraries/dependencies
- Modular projects with independent versioning

Commands:

- Add: git submodule add <repository> <path>
- Initialize: git submodule init
- Update: git submodule update

Characteristics:

- Submodule points to specific commit, not branch
- Requires extra steps when cloning (--recursive)
- Changes in submodule need separate commit

Alternatives include Git subtrees or package managers.