Git & GitHub Interview Q/A

📌 Section 1: Basics

Q1. What is Git?

Answer: Git is a **distributed version control system (DVCS)** that helps track code changes, collaborate with teams, roll back to older versions, and manage branching/merging.

💡 ***Pro Tip:*** Always mention "distributed" — it highlights Git's offline capability.

Q2. What is GitHub?

Answer: GitHub is a **cloud platform** to host Git repositories and collaborate using Pull Requests, Issues, Actions (CI/CD), and project boards.

💡 ***Pro Tip:*** Say Git = tool, GitHub = platform.

Q3. Common Git commands?

Answer:

- **git init** → Initialize a new Git repository
- **git clone <url>** → Copy repository from remote to local
- **git status** → Show modified/untracked files
- **git add .** → Stage all changes
- **git commit -m "msg"** → Save a snapshot with a message
- **git log --oneline** → Show commit history in short form
- **git branch** → List branches
- **git checkout -b <branch>** → Create + switch to a new branch
- **git merge <branch>** → Merge branch into current
- **git push origin main** → Push local commits to GitHub
- **git pull** → Fetch + merge changes from remote
- **git fetch** → Only download changes (no merge)

  💡 ***Pro Tip:*** In interviews, mention git commit -am "msg" as a shortcut for tracked files.

Q4. Difference between Git and GitHub?

Answer:

- Git → Local VCS tool that manages versions of code.
- GitHub → Cloud hosting service that extends Git with collaboration features.
  Interview Edge: Mention alternatives like GitLab, Bitbucket.

Q5. What is a repository?
Answer: A repository is a **project directory managed by Git**, containing code, commit history, and branches.

- **Local repository** → Exists on your machine (git init).
- **Remote repository** → Hosted on GitHub for team collaboration.
  💡 ***Pro Tip:*** Interviewers like when you highlight the difference between local and remote repos.

Q6. What is a commit?
Answer: A commit is a **snapshot of code** at a point in time, identified by a unique SHA hash, author, timestamp, and commit message.
💡 ***Pro Tip:*** A clear commit message helps track why a change was made.

📌 Section 2: Branching & Merging

Q7. What is a branch?
Answer: A branch is a **parallel line of development**. It allows developers to build features without affecting the main branch.
💡 ***Pro Tip:*** Mention how feature branches prevent breaking production code.

Q8. What is a merge conflict? How do you resolve it?
Answer: A merge conflict happens when **two branches modify the same code section**. Git cannot decide which change to keep.
Steps:

1. Open file and check conflict markers.
2. Decide correct changes (or combine).

3. Save → git add → git commit.
   💡 ***Pro Tip:*** Mention that handling conflicts quickly is vital in team projects.

Q9. Difference between git merge and git rebase?
Answer:

- **Merge** → Combines histories, keeps non-linear commits.
- **Rebase** → Replays commits on another branch, making history linear.
  💡 ***Pro Tip:*** Teams often use merge for collaboration, and rebase for personal branches.

Q10. What is a fast-forward merge?
Answer: A fast-forward merge occurs when no new commits exist on the target branch, so Git simply **moves the branch pointer ahead** without creating a merge commit.
💡 ***Pro Tip:*** Fast-forward keeps history simple, but doesn't show that a branch existed.

Q11. What is a detached HEAD?
Answer: Detached HEAD occurs when Git points directly to a commit instead of a branch.

- Commits made here won't belong to any branch unless you create one.
  💡 ***Pro Tip:*** Used for experiments or reviewing older commits.

📌 Section 3: Commands

Q11. Common Git commands?
Answer:

- **git init** → Initialize a new Git repository
- **git clone <url>** → Copy repository from remote to local
- **git status** → Show modified/untracked files
- **git add .** → Stage all changes

- **git commit -m "msg"** → Save a snapshot with a message
- **git log --oneline** → Show commit history in short form
- **git branch** → List branches
- **git checkout -b <branch>** → Create + switch to a new branch
- **git merge <branch>** → Merge branch into current
- **git push origin main** → Push local commits to GitHub
- **git pull** → Fetch + merge changes from remote
- **git fetch** → Only download changes (no merge)
  💡 ***Pro Tip:*** In interviews, mention git commit -am "msg" as a shortcut for tracked files.

Q12. What is the difference between git pull and git fetch?
Answer:

- git fetch downloads commits from remote into remote-tracking branches (like origin/main) but doesn't merge them.
- git pull does fetch + merge, updating your current branch automatically.

Q13. What is git stash?
Answer:
git stash temporarily saves uncommitted changes and resets your working directory.
Example: You are mid-feature but must switch to main to fix a bug. Run git stash, switch branch, fix and commit the bug, then return and restore with git stash pop.

📌 Section 4: Staging & Workflow

Q14. What is git add and why do we need it?
Answer: git add stages changes from the working directory to the staging area. Git uses a **3-stage system**: Working Directory → Staging Area → Repository.

Q15. Do we always need git add . before commit?
Answer: No. For tracked files, you can commit directly with git commit -am "msg". But for new/untracked files, git add is mandatory.

Q16. What is git stash and when is it useful?
Answer: git stash temporarily saves uncommitted work and clears the working directory.
Example: In the middle of a feature, you need to switch to main to fix a bug → run git stash, later use git stash pop to restore.

Q17. What is .gitignore?
Answer: A file that lists files/folders Git should ignore (e.g., logs, build files, environment secrets).

Q18. Difference between git reset, git revert, and git checkout?
Answer:

- **reset** → Moves HEAD pointer back, possibly deleting commits.
- **revert** → Safely creates a new commit that undoes changes.
- **checkout** → Switches branches or restores files.
  💡 ***Pro Tip:*** Say "I prefer revert in teams because it preserves history."

📌 Section 5: Remote Operations

Q19. What is the difference between git pull and git fetch?
Answer:

- **fetch** → Downloads changes into remote-tracking branch (origin/main) but doesn't merge.
- **pull** → Fetch + Merge into local branch.

Q20. What is a remote in Git?
Answer: A remote is a **reference to a hosted repository** (usually on GitHub). Default name = origin.

Q21. How do you push a branch to GitHub?
Answer:

git push origin branch-name

For first time:

git push -u origin branch-name

(-u sets upstream tracking for future pushes).

Q22. What is the difference between fork and clone?
Answer:

- **Fork** → Copies repository to your GitHub account.
- **Clone** → Copies repository to your local machine.

Q23. What is a Pull Request (PR)?
Answer: A PR is a **request to merge code** into another branch on GitHub. It allows code review, discussion, and approval before merging.

📌 Section 6: Tags & Releases

Q24. What is a Git tag?
Answer: A tag is a **label pointing to a specific commit**, often used for releases (v1.0.0).

Q25. Difference between lightweight and annotated tags?
Answer:

- Lightweight → Simple pointer.
- Annotated → Stores metadata like tagger, date, and message (preferred for releases).

Q26. How do you create and push a tag?
Answer:

git tag -a v1.0.0 -m "First release"

git push origin v1.0.0

Q27. How do you delete a tag?
Answer:

- Local: git tag -d v1.0.0
- Remote: git push --delete origin v1.0.0

Q28. How to see commits between two tags?
Answer:

git log v1.0.0..v2.0.0 --oneline

📌 Section 7: Advanced & Best Practices

Q29. What is GitHub Actions?
Answer: GitHub Actions is a **CI/CD automation tool** inside GitHub. It allows workflows like build, test, and deployment triggered on events (push, PR, etc.).

Q30. What is GitHub Flow vs Git Flow?
Answer:

- **GitHub Flow** → Simple strategy: create branch → PR → merge into main.
- **Git Flow** → Structured: uses main, develop, feature, release, hotfix branches.
  💡 ***Pro Tip:*** Mention GitHub Flow for startups, Git Flow for enterprise projects.

Q31. How do you handle large files in GitHub?
Answer: By using **Git LFS (Large File Storage)**, which stores large binaries outside normal Git history.

Q32. What is the difference between main and origin/main?
Answer:

- **main** → Local branch.

- **origin/main** → Remote-tracking branch (state of remote main).

Q33. Best practices with Git in teams?
Answer:

- Write clear commit messages.
- Use branches for new features.
- Keep main protected.
- Resolve merge conflicts early.
- Use PRs for review.
  💡 ***Pro Tip:*** Add “We also use GitHub Actions for CI/CD” to show awareness of modern workflows.