Mock Interview Guide
Git and GitHub

Instructions for Interviewer:

● You are playing the role of **interviewer**. Use this guide as a script.
● Ask each question one at a time. Follow the steps: **Definition → Details → Scenario → Follow-up.**
● If the interviewee struggles, use the **hint**.
● The goal is to keep it conversational and practical. Help the interviewee think and express their learning.
● **colors assigned:** Questions Answers Hint

Freshers - Level
Git and GitHub
(10 Easy DevOps Interview Questions)

---

1. "What is Git?"

✓ Expected Answer: "Git is a distributed version control system that tracks code changes."

Hint: ***"When multiple developers work on the same codebase, this tool helps keep track of who changed what, and when."***

2. "What is a commit in Git?"

✓ Expected Answer: "A commit is a snapshot of your changes saved in the Git history."

Hint: ***"It's like pressing 'save' on your work and adding a message to explain what you changed."***

3. “What is a branch in Git?”

✅ Expected Answer: “A branch is an independent line of development.”

Hint: ***“It lets you try out features or fixes without affecting the main project until you’re ready.”***

4. “How do you check the current branch?”

✅ Expected Answer: “By running git branch.”

Hint: ***“This command lists all branches — and highlights the one you’re currently on with an asterisk”***

5. “What does git clone do?”

✅ Expected Answer: “It copies a remote repo to your local system.”

Hint: ***“If you want to download someone’s project and work on it locally, this is the command you use.”***

6. “What does git push do?”

✅ Expected Answer: “It uploads your local changes to the remote repository.”

Hint: ***“Once your changes are committed locally, use this to share them with others through GitHub or GitLab.”***

7. “What is the difference between git fetch and git pull?”

✅ Expected Answer: “git fetch downloads new data; git pull does that and merges it.”

Hint: ***“One command just checks for updates, the other brings them in and applies them to your current work.”***

8. “What does git status show?”

✅ Expected Answer: “It shows changes staged, unstaged, or untracked.”

Hint: ***“Use this when you’re unsure what Git sees — it gives you a clean summary of your file states.”***

9. “What is the use of .gitignore?”

✅ Expected Answer: “It lists files and folders to ignore from Git tracking.”

Hint: ***“Want to avoid accidentally pushing sensitive or unnecessary files? This file helps prevent that.”***

10. “What is git merge?”

✅ Expected Answer: “It integrates changes from one branch into another.”

Hint: ***“You’ve finished your feature branch and now want it in the main code — this is the command to use.”***

SCENARIO-BASED INTERVIEW QUESTIONS

1.* Ask:** ***"You pushed your code to GitHub, but it's not showing in the pull request. What could be the reason?"

✅ **Expected:** ***Code might have been pushed to a different branch or repository than the one the PR is targeting.***

Hint: ***Check if the right branch and remote are selected.***

2.* Ask:** ***"You're seeing a 'merge conflict' in a pull request. What does it mean?"

✅ **Expected:** ***Conflicts occur when two branches modify the same part of a file and GitHub can't auto-merge.***

Hint: ***Git needs your help when two edits clash.***

3.* Ask:** ***"You committed to GitHub but forgot to add a file. How can you include it in the same commit?"

✅ **Expected:** ***Add the missing file, then run* git commit --amend *and force push.***

Hint: ***Use amend to fix your last commit.***

4.* Ask:** ***"You cloned a GitHub repo but can't push your changes. What might be wrong?"

✅ **Expected:** ***You may not have write access, or the remote URL uses HTTPS without proper credentials.***

Hint: ***Push permissions come from GitHub repo access.***

5. Ask: *"What is the use of a* .gitignore *file in GitHub projects?"*

✅ **Expected:** ***It tells Git which files/folders to ignore and not track (e.g., logs, env files).***

Hint: ***Keeps your repo clean and secure.***

PROJECT-BASED INTERVIEW QUESTIONS

---

1. Ask: *"How would you set up a new GitHub repository for your project and start committing code?"*

✅ **Expected:** ***Create repo on GitHub → clone locally →*** **git init, git remote add, git add, commit,** ***and*** **push.**

Hint: ***Start local → push to remote.***

2. Ask: *"You want to collaborate with a teammate on GitHub. How would you manage the workflow?"*

✅ **Expected:** ***Use branches for features, open pull requests, review code, then merge to main.***

Hint: *Branch → PR → Review → Merge.*

3. **Ask:** *"How can you protect your main branch from direct pushes?"*

✅ **Expected:** ***Enable branch protection rules in GitHub → Require pull requests → Disable force push.***

Hint: ***Only reviewers should allow changes to*** **main.**

4. **Ask:** *"How would you link a GitHub repo with your local machine to keep code in sync?"*

✅ **Expected:** ***Use*** **git remote add origin <repo-url>**, ***then push/pull changes to/from GitHub.***

Hint: ***Your local and GitHub repo must stay connected via remote.***

Medium - Level
Git and GitHub
(DevOps Interview Questions - 1 to 2 Years Experience)

1. “What is the difference between Git and GitHub?”

✅ **Answer:**
Git is a distributed version control system for tracking code changes.
GitHub is a web-based platform that hosts Git repositories and adds collaboration features.

Hint: ***One works locally, the other is for remote sharing and teamwork.***

2. “What is the purpose of branching in Git?”

✅ **Answer:**
Branching allows developers to work on new features or fixes in isolation.
It avoids affecting the main codebase until changes are merged.

Hint: ***You want to try changes without breaking the main app.***

3. “What is a merge conflict and how do you resolve it?”

✅ **Answer:**
A merge conflict occurs when Git can't automatically combine changes from two branches.
You resolve it by editing the conflicting files manually and committing the result.

Hint: *Git shows you where it's confused — you decide what stays.*

4. "How do git fetch and git pull differ?"

✓ **Answer:**
git fetch downloads changes but doesn't merge them.
git pull fetches and merges the remote changes into your local branch.

Hint: *One checks, the other checks and applies.*

5. "What is a rebase in Git?"

✓ **Answer:**
git rebase moves or combines commits from one branch onto another to keep history linear.
It helps in cleaner commit history but must be used carefully.

Hint: *Like replaying your changes on top of another branch.*

6. "How do you revert a specific commit?"

✓ **Answer:**
Use git revert <commit> to create a new commit that undoes the changes.
This is safe because it doesn't rewrite history.

Hint: *Need to undo a bad change, but keep history intact?*

7. "What is a pull request in GitHub?"

✓ **Answer:**

A pull request lets you propose changes to a repository and get them reviewed before merging.
It's a key part of collaboration workflows.

Hint: *A safe way to suggest edits to shared code.*

8. "How do you squash commits in Git?"

✓ Answer:
Use git rebase -i to combine multiple commits into one.
It helps in keeping the commit history clean before merging.

Hint: *Too many tiny commits? Make them one.*

9. "What are Git tags used for?"

✓ Answer:
Tags mark specific points in history, often used to label releases (e.g., v1.0).
They can be lightweight or annotated.

Hint: *You want to remember a specific commit like a release point.*

10. "How do you protect the main branch in GitHub?"

✓ Answer:
Use branch protection rules to enforce checks like code reviews, passing CI, or restrict push access.
It helps maintain code quality and team discipline.

Hint: *Want to prevent force pushes or direct changes?*

SCENARIO-BASED INTERVIEW QUESTIONS

1. Ask: ***“You force-pushed to the*** **main** ***branch and your teammate’s commits are gone. What happened and how can you recover?”***

✅ **Expected:** ***Force push overwrote remote history. Use*** **git reflog** ***or their local branch to recover commits.***

Hint: ***Git doesn’t forget easily — check the reflog.***

2. Ask: ***“A GitHub Action in your repo fails because it can’t access a secret variable. What could be wrong?”***

✅ **Expected:** ***The secret may be missing, misnamed, or not available in forks or pull requests for security reasons.***

Hint: ***Double-check secrets under repo settings — casing matters.***

3. Ask: ***“You have multiple open pull requests that are all failing CI. How would you find if the issue is in the code or the pipeline?”***

✅ **Expected:** ***Check CI logs for each PR. If all fail at the same stage, it's likely a pipeline/config issue.***

Hint: ***Look for patterns in failures across PRs.***

4. **Ask:** ***"Your GitHub repo has hundreds of commits. You want to squash them before merging. How would you do it?"***

✅ **Expected:** ***Use*** **git rebase -i HEAD~n** ***locally, squash commits, then force push. OR squash via GitHub PR option.***

Hint: ***Squash keeps history clean — do it before merging.***

5. **Ask:** ***"You added a GitHub Action to run tests on push, but it never triggers. What might be missing?"***

✅ **Expected:** ***Check if the*** **on: push** ***event is correctly configured in the workflow YAML.***

Hint: ***Even CI needs clear triggers to start.***

PROJECT-BASED INTERVIEW QUESTIONS

---

1. **Ask:** ***"How would you set up a GitHub Action to build and test a Node.js project automatically on each push?"***

✅ **Expected:** ***Create*** **.github/workflows/ci.yml** ***with*** **on: push,** ***use a*** **node** ***runner, then install deps and run*** **npm test.**

Hint: ***CI = trigger + run steps + exit clean.***

2. **Ask:** ***"You want to enforce code review and branch naming rules in a GitHub team. How would you implement this?"***

✅ **Expected:** ***Use branch protection rules and set naming conventions via CI or GitHub apps.***

Hint: ***Rules make your team work better — automate enforcement.***

3. Ask: "How would you automate semantic version tagging for each release in GitHub?"

✅ **Expected:** ***Use GitHub Actions or release scripts that auto-increment version, tag it, and publish a GitHub Release.***

Hint: ***Let the pipeline decide the next version.***

4. Ask: "You want to mirror your GitHub repo to another Git provider like GitLab. How would you do this?"

✅ **Expected:** ***Set up a push mirror in GitHub or use a scheduled sync script with Git remotes (*****git remote add mirror ...*****).***

Hint: ***Mirroring = push to two places at once.***

Hard - Level
Git and GitHub
(DevOps Interview Questions - 3+ Years Experience)

---

1. “What is the difference between git reset, git revert, and git checkout?”

✓ Answer:

reset alters commit history and pointer.

revert creates a new commit to undo a past one.

checkout switches branches or restores files.

> **Hint:** ***One rewrites history, one undoes it cleanly, and one changes context.***

2. “How do you rewrite Git history safely in a shared repo?”

✓ Answer:
Use git rebase -i only on feature branches and communicate with the team before force-pushing.
Avoid rewriting main or shared history.

> **Hint:** ***Rebasing is safe—only if you're the only one touching the branch.***

3. “Explain the Git three-tree architecture.”

✓ **Answer:**
Git uses three trees:

Working Directory (files on disk)

Index (staged changes)

HEAD (last committed snapshot)

Hint: ***From editing to staging to committing — each step has a tree.***

4. “What is a non-fast-forward error and how do you resolve it?”

✓ **Answer:**
It occurs when your local branch is behind the remote.
Resolve by pulling the latest changes (git pull --rebase) before pushing

.

Hint: ***Remote has moved ahead — you need to catch up before pushing.***

5. “What is a bare Git repository and why is it used?”

✓ **Answer:**
A bare repository contains no working directory.
It’s used as a central repo on a server for collaboration.

Hint: ***This repo isn’t for editing — only for pushing and pulling.***

6. “What’s the difference between merge and rebase in team workflows?”

✓ **Answer:**

merge preserves history and shows branch structure.
rebase rewrites history to be linear and clean.
Use rebase for feature branches and merge for collaborative integration.

Hint: ***Do you want full historical truth, or clean single-line history?***

7. “How do you recover a commit deleted by git reset --hard?”

✓ Answer:
Use git reflog to find the lost commit’s reference and checkout or cherry-pick it.
Reflog keeps a record of all recent HEAD changes.

Hint: ***You thought it was gone, but Git remembers everything—temporarily.***

8. “What is the difference between origin/main and main?”

✓ Answer:
main is your local branch; origin/main is your last fetched copy of the remote.
They differ until you git fetch or git pull.

Hint: ***Which one is updated depends on when you last synced.***

9. “How do Git hooks work, and how have you used them?”

✓ Answer:
Hooks are shell scripts triggered by Git events (e.g., pre-commit, post-merge).
Used for linting, enforcing commit messages, or CI triggers.

Hint: *You want actions to run automatically at key points in Git workflow.*

10. "How would you manage a large Git monorepo used by multiple teams?"

✅ **Answer:**
Use clear directory structure, enforce code ownership, CI rules, and optionally sparse-checkout.
Split responsibilities and automate change detection per subproject.

Hint: *Think: performance, clarity, and team boundaries inside one repo.*

SCENARIO-BASED INTERVIEW QUESTIONS

---

1. **Ask:** *"You discovered that secret keys were accidentally pushed to GitHub. What are the immediate steps you'd take?"*

✅ **Expected:** ***Remove keys, rotate them immediately, force push a clean commit history, and invalidate exposed credentials.***

Hint: *History never forgets — you have to scrub and revoke.*

2. **Ask:** *"You're contributing to an open-source GitHub project using forks, but your PR isn't passing CI due to missing secrets. Why?"*

✅ **Expected:** ***Secrets aren't available to workflows triggered from forks for security. CI needs to run from the main repo or manually.***

Hint: *GitHub protects secrets on external contributions.*

3. **Ask:** ***“You want to detect and prevent large files (>100MB) from being pushed to GitHub. How would you enforce this?”***

✅ **Expected:** ***Use Git hooks or GitHub Actions to reject large files. You can also use Git LFS for large assets.***

Hint: ***Not all files belong in Git — track smart.***

4. **Ask:** ***“How would you investigate who deleted a branch or a tag in your GitHub repository?”***

✅ **Expected:** ***Use the GitHub Audit Log (in GitHub Enterprise or org settings) to trace user actions.***

Hint: ***Forensic tracking? Audit logs are gold.***

5. **Ask:** ***“You noticed your GitHub Action started failing after a minor YAML edit. What's your debugging approach?”***

✅ **Expected:** ***Use* actions/toolkit *logging, set* ACTIONS_STEP_DEBUG=true, *and check indentation or syntax errors in YAML.***

Hint: ***YAML breaks silently — debug visibility matters.***

PROJECT-BASED INTERVIEW QUESTIONS

---

1. **Ask:** ***“Design a GitHub Actions workflow that builds, tests, and deploys a Python app to AWS on merge to main.”***

✅ **Expected:** *Use stages: checkout → install deps → run tests → deploy to AWS via CLI or CDK. Add* **on: push** *to* **main.**

Hint: *Deployment should only trigger after validation.*

2. Ask: *"How would you build an internal GitHub App to validate commit messages before allowing a PR merge?"*

✅ **Expected:** *Use GitHub App + REST API to analyze commit messages in PR, and set a status check to block merge if invalid.*

Hint: *You can build your own reviewers using GitHub APIs.*

3. Ask: *"You're asked to implement branch naming enforcement (e.g.,* **feature/*, bugfix/***). How would you do this in GitHub?"*

✅ **Expected:** *Use GitHub Actions or pre-push Git hooks to validate branch name patterns.*

Hint: *Automation starts with naming rules.*

4. Ask: *"You want to auto-close stale issues and PRs after 30 days of inactivity. How would you automate this?"*

✅ **Expected:** *Use GitHub's* **actions/stale** *action to auto-label and close old issues/PRs.*

Hint: *Let bots keep your backlog clean.*