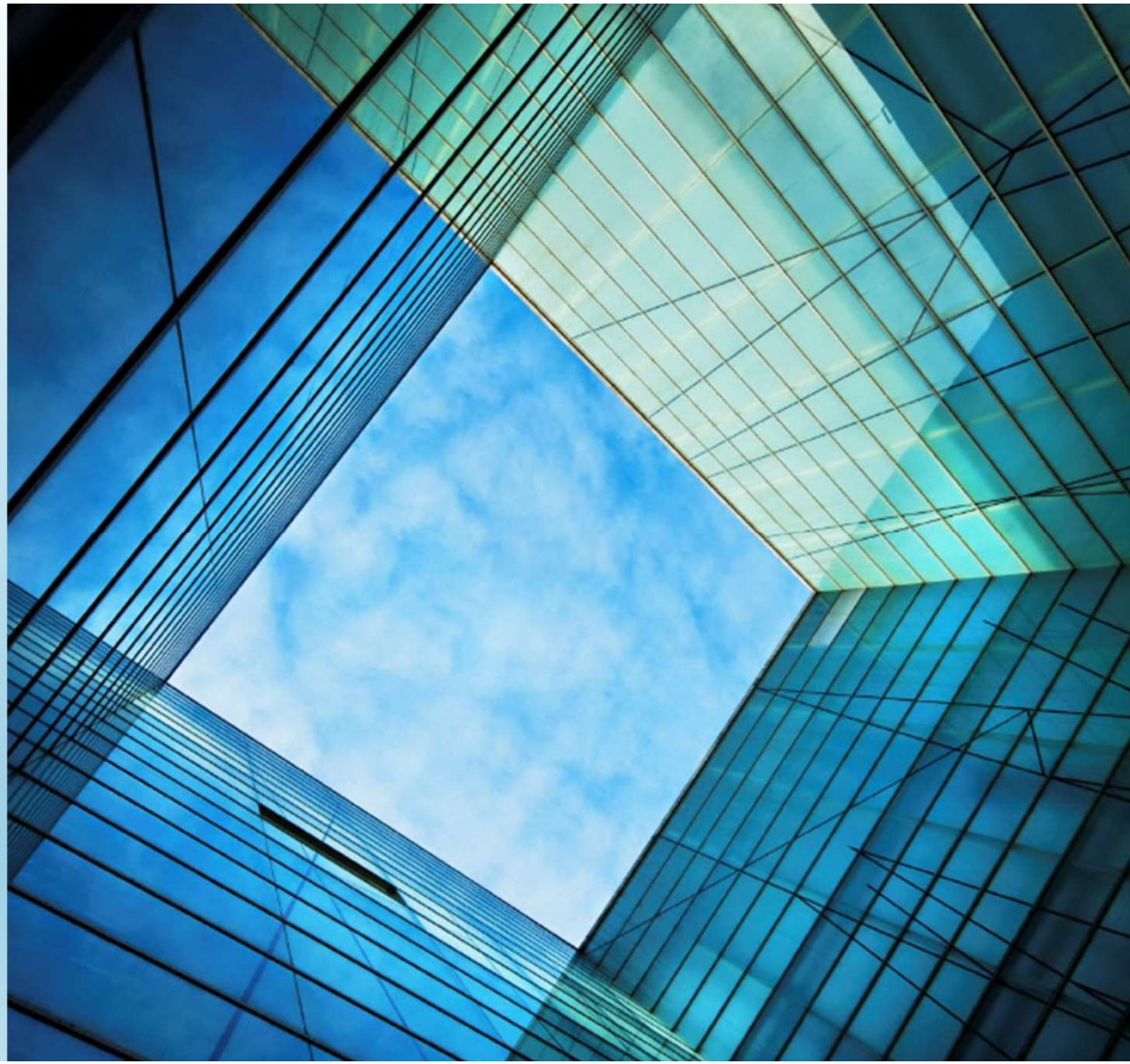
# GIT and GITHUB Interview Questions



- Q: Can you explain the different branching strategies you have used in Git?
- A: Certainly! In my experience, there are several common branching strategies that teams can adopt based on their needs. These include Git Flow and GitHub Flow among others.
- Git Flow is a highly structured method that works well for projects with scheduled release cycles. It has two long-lived branches: master for production and develop for integrating features. Each new feature has its own branch and is merged into develop when it's ready for integration. For a release, we fork a release branch off of develop. Once the release is ready, it is merged into master and develop. For hotfixes, we create a hotfix branch directly off of master.
- GitHub Flow is simpler and ideal for continuous delivery environments. The master branch is always deployable. For every new feature or bugfix, a new branch is created off master. Once the changes are ready and tested, a pull request is opened. If it passes code review, the changes are merged back into master and the feature branch is deleted. After merging, the master branch is deployed to production.

# GIT and GITHUB Interview Questions

- Q: How would you undo the most recent commit in Git?
- A: You can use the command **git reset --soft HEAD~1** to undo the most recent commit, keeping the changes in the staging area, or **git reset --hard HEAD~1** to discard the changes completely.

- Q: How do you deal with a merge conflict in Git?
- A: When a merge conflict happens, Git will indicate the conflicted files. You need to open those files and look for the conflict markers (<<<<<<<, =======, >>>>>>>). The changes from the current branch are above the =======, and the changes from the branch being merged are below. Edit the file to resolve the conflict, then add the file with git add, and commit it with git commit.

# GIT and GITHUB Interview Questions

- Q: Explain the difference between git fetch and git pull.
- A: git fetch only downloads new data from a remote repository, but it doesn't integrate any of this new data into your working files. On the other hand, git pull updates your current HEAD branch with the latest changes from the remote server. This means git pull is equivalent to git fetch followed by git merge.

- Q: Can you describe a typical Git workflow you would use in a collaborative project?
- A: A common workflow is the feature branch workflow. This involves creating a new branch whenever you want to work on a new feature. The master branch should never contain broken code, it is always production-ready. All development is done in branches and then merged into the master branch once the feature is complete and tested.

# GIT and GITHUB Interview Questions

- Q: What are the steps to perform a rebase in Git?
- A: A rebase can be performed using the following steps:
  - Switch to the branch you want to rebase: git checkout feature-branch
  - Start the rebase process: git rebase master
  - If there are any conflicts, resolve them. After resolving conflicts, use git add . to stage the resolved files, then continue the rebase with git rebase --continue.
  - If you want to abort the process at any time, use git rebase --abort.

- Q: How do you go about resolving a binary file conflict?
- A: Binary files cannot be merged like text files, so you have to decide which version to keep. You can do this with git checkout --ours filename or git checkout --theirs filename and then commit the resolved file.

# GIT and GITHUB Interview Questions

**Git Rebase**

- Git rebase is a command that can be used to integrate changes from one branch into another. It is an alternative to the better known git merge. Both of these commands are designed to integrate changes from one branch into another.
- Rebase is a bit different because instead of merging the branches, it actually moves or combines the changes via a series of patch. This makes for a more streamlined, linear project history.
- Scenario for Rebase:
- Let's assume you're working in a DevOps team. You have a development branch where all the development work happens. Once the development is done, it's merged into the main branch, which is then deployed to production.
- However, multiple developers working on the development branch can cause it to quickly get out of sync with the main branch, especially if other teams are also merging their changes into main.
- To keep development up-to-date with main, you can use the git rebase command. This will allow you to apply the changes from main onto development as if those changes happened after all the changes on development. Here's how you can do it:
- # switch to development branch
  - $ git checkout development
- # rebase the development branch onto main
  - $ git rebase main

# REBASE EXAMPLE

- Q: What is the function of git cherry-pick?
- A: git cherry-pick is used to apply the changes from an existing commit to the current branch. It basically generates a new commit with a different commit hash.

- Q: Could you explain how to squash commits in Git?
- A: Squashing is the process of combining several commits into a single one. This is usually done in the context of cleaning up a feature branch before merging into main. This is typically performed through an interactive rebase, for example, **git rebase -i HEAD~n** (where 'n' is the number of commits to squash from the current HEAD).

# GIT and GITHUB Interview Questions

# Cherry pick

- git cherry-pick is a command in Git that allows you to take a commit from one branch and apply it onto another branch. It's like saying, "I want that specific change they made in that commit over there on my branch here."
- Here is a simple breakdown:
- You've worked on a feature in a branch (let's call it "branch A") and made several commits to record your changes.
- Meanwhile, you also worked on another branch (let's call it "branch B").
- You realize that one of the changes (commits) you made on branch A would be really helpful on branch B.
- Instead of manually recreating that change on branch B, you can use **git cherry-pick <commit-hash>** to copy that specific commit from branch A and apply it onto branch B.
- It's important to remember that cherry-pick applies the changes made in specific commits and not the entire set of changes in a branch. It's like picking just the cherry (commit) you want, instead of taking the entire sundae (all commits on the branch).

# Squashing

- "Squashing" in Git is a technique used to clean up your commit history.
- Here's a simple way to understand it:
- Imagine that you're working on a big project and you've made a bunch of changes. Each time you make a set of related changes, you "commit" them, which is like saving your work in a special way that Git can keep track of. After a few days, you might have made 10 commits.
- But maybe you realize that all these changes were really just about one thing, like "adding a new feature" or "fixing a bug". It's a bit messy to have all these separate commits about the same thing. This is where "squashing" comes in.
- "Squashing" is like taking a stack of papers, each with different parts of a story written on them, and combining them all onto one page. With squashing in Git, you take all those different commits and combine them into a single commit. This makes your project's history cleaner and easier to understand.
- In short, squashing in Git is all about tidying up your project's history by combining multiple commits into one.

**Git Cherry-Pick**

- Git cherry-pick allows a developer to select specific commits from one branch and apply them onto another branch. It's a way of applying some commits from one branch onto another.
- Scenario for Cherry-Pick:
- Continuing with the DevOps scenario, let's say you have made several commits to your development branch that fix a critical bug. However, this branch also contains a lot of new features that aren't ready for production yet.
- Instead of merging all the changes from development into main, you can use git cherry-pick to apply only the bug-fix commits to main. This allows you to fix the bug in production without deploying untested features.
- First, you need to find the hash of the bug-fix commits using git log. Once you have the hashes, you can checkout to the main branch and apply the commits using git cherry-pick:
- # switch to main branch
  - $ git checkout main
- # cherry-pick the commit
  - $ git cherry-pick commit_hash

# CHERRY PICK EXAMPLE

- Q: What's your approach to secure a Git repository?
- A: This could involve measures such as using SSH keys for authentication, regularly updating and auditing contributor access, implementing commit signing to verify the authenticity of commits, using .gitignore to prevent committing sensitive data, and using security tools to automatically scan your repositories for security vulnerabilities or secrets.

- Q: Can you describe the different types of Git hooks?
- A: Git hooks are scripts that run automatically every time a particular event occurs in a Git repository. They are used for automation of workflow tasks. Examples of hooks include pre-commit (runs before commit), post-commit (runs after commit), pre-receive (runs on the remote side before acknowledging a push), and others.

# GIT and GITHUB Interview Questions

- Q: Explain the difference between a Git fork and a Git clone.
- A: A fork is a remote, server-side copy of a repository, distinct from the original. A clone is a local copy of some remote repository. When you clone, you are making a copy of the complete repository history, but with a fork, you are making a server-side copy which can be tracked separately.

- Q: How would you handle large files or large amounts of binary data in a Git repository?
- A: Git is not designed to handle large files or large volumes of binary data. The Git LFS (Large File Storage) extension can be used for versioning large files while keeping them out of the actual Git repository.

# GIT and GITHUB Interview Questions

- Q: Explain the use and benefit of a Git submodule.
- A: Submodules allow you to keep another Git repository in a subdirectory of your repository. This lets you clone another repository into your project and keep your commits separate.

- Q: How do you enforce code quality standards and reviews in GitHub?
- A: This can be accomplished through a combination of tactics, including pull request templates, code review procedures, continuous integration checks, and using GitHub's built-in "protected branches" feature to require certain checks pass before merging.

# GIT and GITHUB Interview Questions

- Q: What are GitHub Actions and how have you used them in a project?
- A: GitHub Actions are a CI/CD and general automation feature of GitHub, allowing you to run scripts (known as "workflows") in reaction to Git events like push, pull request, and more. They can be used for a wide variety of purposes, from testing code to deploying applications.

- Q: What are some strategies you might use to keep the history of a Git repository clean?
- A: Strategies include interactive rebase to squash or fixup commits, avoiding unnecessary merge commits with rebasing, using git commit --amend for fixing recent mistakes, and using descriptive commit messages to clearly articulate changes.

# GIT and GITHUB Interview Questions

- Q: How have you used GitHub or other online Git platforms in a Continuous Integration / Continuous Delivery (CI/CD) pipeline?
- A: The answer will depend on the individual's experience. They may discuss using GitHub Actions to run tests and deploy code, setting up Jenkins to use Git repositories, or integrating with other tools like Travis CI, CircleCI, etc.

- Q: What do you consider best practices for managing branches in Git?
- A:  Always create a new branch for new features or fixes, regularly merge the main branch into your branch to keep it up-to-date, delete branches after their changes have been merged, and use meaningful branch names.

# GIT and GITHUB Interview Questions

- Q: How can one revert a pull request in GitHub?
- A: To revert a pull request in GitHub, you can use the "Revert" button in the GitHub interface. This creates a new pull request that undoes all changes made in the original pull request.

- Q: Explain how you would set up a Git repository to run code checks automatically when someone pushes a change.
- A: This can be done using Git hooks, specifically the pre-receive hook on the server side, or the pre-push hook on the client side. You can also use CI/CD pipelines with GitHub Actions, Jenkins, or other tools to run checks automatically on push.

# GIT and GITHUB Interview Questions

- Q: How do you synchronize a forked GitHub repository with the original repository?
- A: This can be done by adding the original repository as a remote (git remote add upstream <repo-url>), fetching the updates (git fetch upstream), and then merging the updates into your fork (git merge upstream/main).

- Q: What are some advanced features of GitHub that you find particularly useful and why?
- A: The answer to this will depend on the individual's experience, but may include features like GitHub Actions, code scanning, security advisories, pull request reviews, the dependency graph, and others.

# GIT and GITHUB Interview Questions