**Git Interview Questions for Automation Testers (2 Years Experience)**

**I. Basic Git Concepts & Commands**

**1. Q: What is Git, and why is it important for an automation tester?**

* **A:** Git is a distributed version control system (DVCS) that tracks changes in source code and other files during software development. For an automation tester, it's crucial for:
  + **Collaboration:** Working with other testers and developers on the same test automation framework.
  + **Version Control:** Keeping track of changes in test scripts, page object models, utility files, and configurations.
  + **Rollback:** Easily reverting to previous stable versions of test code if a new change breaks something.
  + **Branching:** Developing new test features or bug fixes in isolation without affecting the main codebase.
  + **Code Review:** Facilitating the review of test automation code before merging.

**2. Q: Explain the difference between Git and GitHub.**

* **A:**
  + **Git:** Is the *version control system* itself. It's the software you install locally on your machine to manage your code's history. It allows you to commit, branch, merge, and track changes.
  + **GitHub:** Is a *web-based hosting service* for Git repositories. It provides a centralized platform for teams to collaborate, share code, manage pull requests, and host their Git projects remotely. Think of Git as the engine and GitHub as the garage where you store and share your cars (code).

**3. Q: What are the three states of files in Git?**

* **A:**
  + **Working Directory (Modified):** The current state of your files on your local machine. When you make changes, they are in this state.
  + **Staging Area (Staged):** A temporary area where you prepare changes before committing them. You use git add to move files from the working directory to the staging area.
  + **Git Repository (Committed):** The permanent storage where your changes are saved as commits. Once changes are committed, they are part of the project's history.

**4. Q: How do you initialize a new Git repository?**

* **A:** You navigate to the root directory of your project in the terminal/command prompt and run:

Bash

git init

This creates a hidden .git directory, which is where Git stores all the history and configuration for your repository.

**5. Q: What is the purpose of git add . and git commit -m "message"?**

* **A:**
  + git add .: This command stages *all* modified and new files in your current directory (and subdirectories) for the next commit. It moves them from the "Working Directory" to the "Staging Area."
  + git commit -m "message": This command takes all the changes currently in the "Staging Area" and permanently records them in the Git repository as a new commit. The -m flag allows you to provide a concise commit message describing the changes.

**6. Q: How do you check the status of your Git repository (what files are modified, staged, etc.)?**

* **A:** You use the command:

Bash

git status

This command provides information on modified files, untracked files, and staged changes, helping you understand the current state of your repository.

**7. Q: How do you view the commit history of a repository?**

* **A:** You use the command:

Bash

git log

This displays a list of commits in reverse chronological order, showing the commit hash, author, date, and commit message.

**II. Branching & Merging**

**8. Q: What is a Git branch, and why is it important for an automation tester?**

* **A:** A Git branch is essentially a separate line of development. It allows you to work on new features, bug fixes, or experimental test cases without affecting the main or stable version of your test automation framework.
* **Importance for Automation Testers:**
  + **Isolation:** Develop new test scripts or enhance existing ones without breaking the main automation suite.
  + **Parallel Development:** Multiple testers can work on different features simultaneously.
  + **Experimentation:** Try out new approaches or libraries without risking the main codebase.
  + **Hotfixes:** Quickly fix urgent issues on a separate branch while main development continues.

**9. Q: How do you create a new branch and switch to it?**

* **A:**
  + Create and switch (shorthand):

Bash

git checkout -b <branch-name>

* + Create, then switch:

Bash

git branch <branch-name>

git checkout <branch-name>

**10. Q: How do you switch between branches?** \* **A:** bash git checkout <branch-name>

**11. Q: How do you merge changes from one branch into another?** \* **A:** 1. First, switch to the branch you want to merge *into* (e.g., main): bash git checkout main 2. Then, merge the other branch into the current branch: bash git merge <branch-to-merge> This applies the changes from <branch-to-merge> to main.

**12. Q: What is a merge conflict, and how do you resolve it?** \* **A:** A merge conflict occurs when Git cannot automatically reconcile differences between two branches being merged, typically because the same lines of code have been modified differently in both branches. \* **Resolution Steps:** 1. Git will notify you of the conflict. 2. Use git status to see which files have conflicts. 3. Open the conflicted files. Git will mark the conflicting sections with <<<<<<<, =======, and >>>>>>>. 4. Manually edit the file to choose which changes to keep, or combine them as desired. 5. Remove the conflict markers (<<<<<<<, =======, >>>>>>>). 6. Stage the resolved file: git add <conflicted-file> 7. Commit the merge: git commit -m "Resolved merge conflict"

**III. Remote Repositories & Collaboration**

**13. Q: How do you connect your local repository to a remote repository (e.g., GitHub)?** \* **A:** 1. Add the remote URL: bash git remote add origin <remote-repository-url> (origin is a common alias for the primary remote repository) 2. Push your local changes to the remote: bash git push -u origin main (or master) The -u (or --set-upstream) sets the upstream branch, so future git push and git pull commands don't require specifying origin main.

**14. Q: What is the difference between git pull and git fetch?** \* **A:** \* git fetch: Downloads changes from the remote repository to your local repository, but *does not* automatically merge them into your current working branch. It updates your remote-tracking branches (e.g., origin/main). You can then inspect the changes before merging. \* git pull: Is a shorthand for git fetch followed by git merge. It fetches changes from the remote and then automatically merges them into your current local branch. Use with caution, as it can sometimes lead to immediate merge conflicts.

**15. Q: How do you push your local changes to the remote repository?** \* **A:** bash git push origin <branch-name> If you've set an upstream branch (e.g., git push -u origin main), you can often just use git push.

**16. Q: What is a Pull Request (PR) or Merge Request (MR), and why is it important in automation testing?** \* **A:** A Pull Request (GitHub terminology) or Merge Request (GitLab/Bitbucket terminology) is a formal request to merge changes from one branch (typically a feature or fix branch) into another (e.g., main). \* **Importance for Automation Testers:** \* **Code Review:** Allows teammates (other testers, developers, leads) to review your test automation code (test scripts, framework changes, locators) before it's merged into the main codebase. This helps catch bugs, improve code quality, ensure adherence to coding standards, and share knowledge. \* **Collaboration:** Facilitates discussion and feedback on new test features. \* **Quality Gate:** Acts as a quality gate, ensuring that new automation code is stable and functional before impacting the main automation suite. \* **Documentation:** PRs often include descriptions of the changes, linked issues, and test results, serving as valuable documentation.

**IV. Advanced Git Concepts (Relevant for Testers)**

**17. Q: When would you use git stash?** \* **A:** git stash temporarily saves your uncommitted changes (both staged and unstaged) in your working directory, allowing you to switch branches or perform other operations without committing incomplete work. \* **Scenario for Automation Testers:** You're working on a new test case, but an urgent bug requires you to switch to another branch to write a hotfix. You can git stash your current work, switch branches, fix the bug, commit, and then switch back to your original branch and git stash pop to reapply your test case changes.

**18. Q: How do you revert a previous commit in Git?** \* **A:** You use git revert <commit-hash>. This creates a *new* commit that undoes the changes introduced by the specified commit, without rewriting history. This is safer than git reset for shared repositories. \* **Scenario for Automation Testers:** A recent commit broke several existing automation tests. Instead of trying to fix it immediately, you might revert that specific commit to get the automation suite back to a stable state, allowing you to investigate the issue separately.

**19. Q: Explain the concept of HEAD in Git.** \* **A:** HEAD is a symbolic reference to the *current commit* you are on. It typically points to the tip of your current branch. When you switch branches or make a new commit, HEAD moves. HEAD can also be detached, meaning it points directly to a commit rather than a branch (e.g., when checking out a specific commit for inspection).

**20. Q: You made some changes, committed them locally, but realized you made a mistake in the commit message. How do you fix it without creating a new commit?** \* **A:** You use git commit --amend. This command allows you to change the message of the most recent commit. If you also want to add more changes to that commit, stage them *before* running git commit --amend.

**21. Q: How would you use Git in your day-to-day automation testing activities? Give specific examples.** \* **A:** \* **Starting New Test Feature:** git checkout -b feature/new-login-tests \* **Developing & Committing:** Write test scripts, POMs, etc. git add ., git commit -m "Add new login test cases and page objects" \* **Getting Latest Code:** git pull origin main before starting work to ensure I have the most up-to-date framework. \* **Creating Hotfix:** git checkout -b hotfix/failed-checkout-test main (from main) then fix, commit, and push. \* **Reviewing Test Code:** Participating in PR reviews for other testers' or developers' automation code. \* **Reverting Bad Changes:** If a recently merged test change breaks a large part of the suite, git revert <commit-hash> to quickly stabilize. \* **Stashing for Interruptions:** git stash when a higher-priority task comes up, then git stash pop later. \* **Synchronizing with Remote:** Regularly git push my feature branch to GitHub for backup and visibility.

**Tips for the Interview:**

* **Be clear and concise:** Avoid jargon where possible, or explain it.
* **Provide real-world examples:** Relate Git concepts to scenarios you'd encounter as an automation tester.
* **Demonstrate practical knowledge:** Show you can actually use the commands, not just define them.
* **Ask clarifying questions:** If a question is unclear, don't hesitate to ask for clarification.
* **Mention your workflow:** Briefly describe how you integrate Git into your daily automation testing workflow.