## Git Interview Questions

I really liked working with Git. Git plays a vital role in many organizations to achieve DevOps and is a must know technology. This reason drives me to prepare you for the most frequently asked Git interview questions.

After a lot of research and discussion with many DevOps experts who have above 10 years of experience in their domain and are frequently taking interviews as well, I have collected the below set of questions. Curious to know more about Git ***check out this Git blog series*** (https://www.edureka.co/blog/what-is-git/).

This Git Interview Questions blog is a part of parent blog ***DevOps Interview Questions*** (https://www.edureka.co/blog/interview-questions/top-devops-interview-questions-2016/). It includes all the DevOps Stages.

First question in this Git Interview Questions blog has to be:

**Q1. What is the difference between Git and SVN?**

**Git vs SVN**

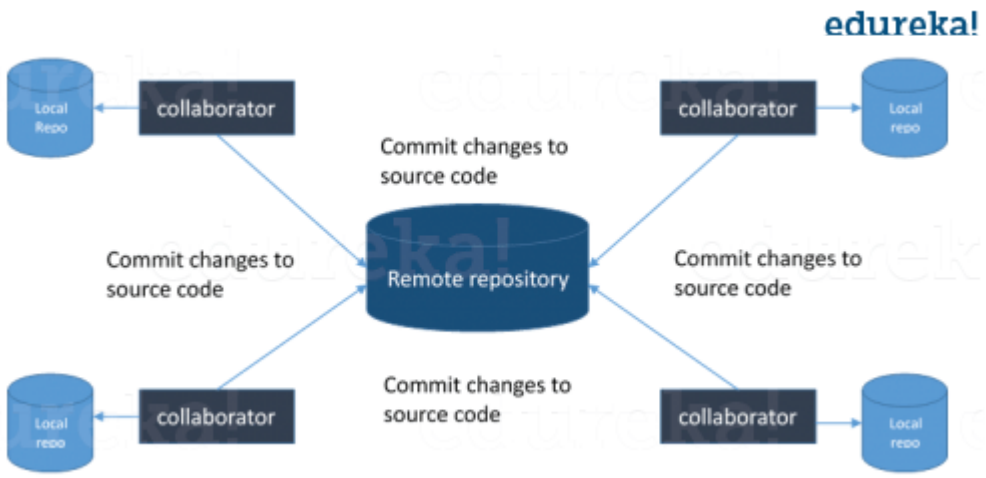
| Git | SVN |
|---|---|
| 1. Git is a Decentralized Version Control tool | 1. SVN is a  Centralized Version Control tool |

| | |
|---|---|
| 2. It belongs to the 3rd generation of Version Control tools | 2. It belongs to the 2nd generation of Version Control tools |
| 3. Clients can clone entire repositories on their local systems | 3. Version history is stored on a server-side repository |
| 4. Commits are possible even if offline | 4. Only online commits are allowed |
| 5. Push/pull operations are faster | 5. Push/pull operations are slower |
| 6. Works are shared automatically by commit | 6. Nothing is shared automatically |

**Q2. What is Git?**

I will suggest you to attempt this question by first telling about the architecture of git as shown in the below diagram just try to explain the diagram by saying:

- Git is a Distributed Version Control system (DVCS). It can track changes to a file and allows you to revert back to any particular change.
- Its distributed architecture provides many advantages over other Version Control Systems (VCS) like SVN one major advantage is that it does not rely on a central server to store all the versions of a project's files.
- Instead, every developer "clones" a copy of a repository I have shown in the diagram with "Local repository" and has the full history of the project on his hard drive so when there is a server outage all you need for recovery is one of your teammate's local Git repository.
- There is a central cloud repository as well where developers can commit changes and share it with other teammates as you can see in the diagram where all collaborators are commiting changes "Remote repository".




*Now, the next set of Git interview questions will test your experience with Git:*

**Q3. What is the command to write a commit message in Git?**

Answer to this is pretty straightforward.

Command that is used to write a commit message is "**git commit -a**".

Now explain about -a flag by saying -a on the command line instructs git to commit the new content of all tracked files that have been modified. Also mention you can use "**git add<file>**"

before git commit -a if new files need to be committed for the first time.

**Q4. What is 'bare repository' in Git?**

You are expected to tell the difference between a "working directory" and "bare repository".

A "bare" repository in Git just contains the version control information and no working files (no tree) and it doesn't contain the special .git sub-directory. Instead, it contains all the contents of the .git sub-directory directly in the main directory itself, where as working directory consist of:

1. A .git subdirectory with all the Git related revision history of your repo.
2. A working tree, or checked out copies of your project files.

**Q5. What language is used in Git?**

Instead of just telling the name of the language, you need to tell the reason for using it as well. I will suggest you to answer this by saying:

Git uses 'C' language. GIT is fast, and 'C' language makes this possible by reducing the overhead of run times associated with high level languages.



**Q6. In Git how do you revert a commit that has already been pushed and made public?**

There can be two answers to this question and make sure that you include both because any of the below options can be used depending on the situation:

- Remove or fix the bad file in a new commit and push it to the remote repository. This is the most natural way to fix an error. Once you have made necessary changes to the file, commit it to the remote repository for that I will use
  **git commit -m "commit message"**
- Create a new commit that undoes all changes that were made in the bad commit.to do this I will use a command
  **git revert <name of bad commit>**

**Q7. What is the difference between git pull and git fetch?**

Git pull command pulls new changes or commits from a particular branch from your central repository and updates your target branch in your local repository.

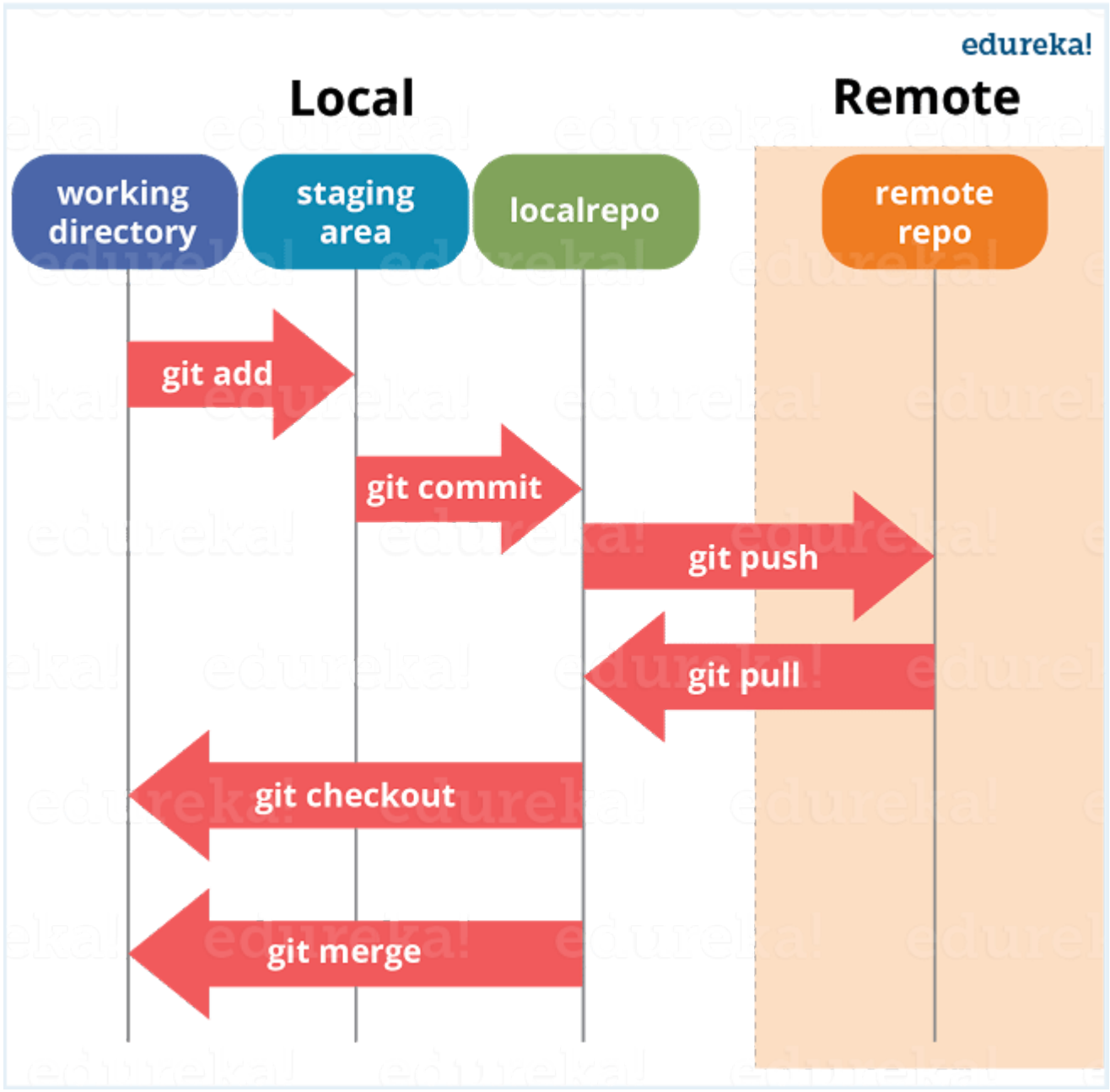
Git fetch is also used for the same purpose but it works in a slightly different way. When you perform a git fetch, it pulls all new commits from the desired branch and stores it in a new branch in your local repository. If you want to reflect these changes in your target branch, git fetch must be followed with a git merge. Your target branch will only be updated after merging the target branch and fetched branch. Just to make it easy for you, remember the equation below:

Git pull = git fetch + git merge

**Q8. What is 'staging area' or 'index' in Git?**

For this answer try to explain the below diagram as you can see:

That before completing the commits, it can be formatted and reviewed in an intermediate area known as 'Staging Area' or 'Index'. From the diagram it is evident that every change is first verified in the staging area I have termed it as "stage file" and then that change is committed to the repository.




*If your interviewer has good knowledge on Git he/she will dig in deep, so the next set of Git interview questions will be more challenging.*

**Q9. What is Git stash?**

According to me you should first explain the need for Git stash.

Often, when you've been working on part of your project, things are in a messy state and you want to switch branches for sometime to work on something else. The problem is, you don't want to do a commit of half-done work just so you can get back to this point later. The answer to this issue is Git stash.

Now explain what is Git stash.

Stashing takes your working directory that is, your modified tracked files and staged changes and saves it on a stack of unfinished changes that you can reapply at any time.

**Q10. What is Git stash drop?**

Begin this answer by saying for what purpose we use Git 'stash drop'.

Git 'stash drop' command is used to remove the stashed item. It will remove the last added stash item by default, and it can also remove a specific item if you include it as an argument.

Now give an example.

If you want to remove a particular stash item from the list of stashed items you can use the below commands:

**git stash list:** It will display the list of stashed items like:
stash@{0}: WIP on master: 049d078 added the index file
stash@{1}: WIP on master: c264051 Revert "added file_size"
stash@{2}: WIP on master: 21d80a5 added number to log

If you want to remove an item named stash@{0} use command **git stash drop stash@{0}**.

**Q11. How do you find a list of files that has changed in a particular commit?**

For this answer instead of just telling the command, explain what exactly this command will do.

To get a list files that has changed in a particular commit use the below command:

**git diff-tree -r {hash}**

Given the commit hash, this will list all the files that were changed or added in that commit. The -r flag makes the command list individual files, rather than collapsing them into root directory names only.

You can also include the below mentioned point, although it is totally optional but will help in impressing the interviewer.

The output will also include some extra information, which can be easily suppressed by including two flags:

**git diff-tree –no-commit-id –name-only -r {hash}**

Here –no-commit-id will suppress the commit hashes from appearing in the output, and –name-only will only print the file names, instead of their paths.

**Q12. What is the function of 'git config'?**

First tell why we need '**git config**'.

Git uses your username to associate commits with an identity. The git config command can be used to change your Git configuration, including your username.

Now explain with an example.

Suppose you want to give a username and email id to associate commit with an identity so that you can know who has made a particular commit. For that I will use:

**git config –global user.name "Your Name":** This command will add username.
**git config –global user.email "Your E-mail Address":** This command will add email id.

**Q13. What does commit object contains?**

Commit object contains the following components, you should mention all the three points present below:

- A set of files, representing the state of a project at a given point of time
- Reference to parent commit objects
- An SHAI name, a 40 character string that uniquely identifies the commit object.

**Q14. How can you create a repository in Git?**

This is probably the most frequently asked questions and answer to this is really simple.

To create a repository, create a directory for the project if it does not exist, then run command "**git init**". By running this command .git directory will be created in the project directory.

**Q15. How do you squash last N commits into a single commit?**

There are two options to squash last N commits into a single commit include both of the below mentioned options in your answer:

- If you want to write the new commit message from scratch use the following command
  **git reset –soft HEAD~N &&**
  **git commit**
- If you want to start editing the new commit message with a concatenation of the existing commit messages then you need to extract those messages and pass them to Git commit for that I will use
  **git reset –soft HEAD~N &&**
  **git commit –edit -m"$(git log –format=%B –reverse .HEAD@{N} (mailto:HEAD..HEAD@%7b1%7d))"**

**Q16. What is Git bisect? How can you use it to determine the source of a (regression)**

**bug?**

I will suggest you to first give a small definition of Git bisect.

Git bisect is used to find the commit that introduced a bug by using binary search. Command for Git bisect is

**git bisect <subcommand> <options>**

Now since you have mentioned the command above explain them what this command will do.

This command uses a binary search algorithm to find which commit in your project's history introduced a bug. You use it by first telling it a "bad" commit that is known to contain the bug, and a "good" commit that is known to be before the bug was introduced. Then Git bisect picks a commit between those two endpoints and asks you whether the selected commit is "good" or "bad". It continues narrowing down the range until it finds the exact commit that introduced the change.

**Q17. How do you configure a Git repository to run code sanity checking tools right before making commits, and preventing them if the test fails?**

I will suggest you to first give a small introduction to sanity checking.

A sanity or smoke test determines whether it is possible and reasonable to continue testing.

Now explain how to achieve this.

This can be done with a simple script related to the pre-commit hook of the repository. The pre-commit hook is triggered right before a commit is made, even before you are required to enter a commit message. In this script one can run other tools, such as linters and perform sanity checks on the changes being committed into the repository.

Finally, give an example, you can refer the below script:

```
#!/bin/sh
files=$(git diff –cached –name-only –diff-filter=ACM | grep '.go$')
if [ -z files ]; then
exit 0
fi
unfmtd=$(gofmt -l $files)
if [ -z unfmtd ]; then
exit 0
fi
echo "Some .go files are not fmt'd"
exit 1
```

This script checks to see if any .go file that is about to be committed needs to be passed through

the standard Go source code formatting tool gofmt. By exiting with a non-zero status, the script effectively prevents the commit from being applied to the repository.

*The Interviewer has not started asking questions on branching yet, so the next set of Git interview questions will be dealing with branching in Git.*

**Q18. Describe branching strategies you have used?**

This question is asked to test your branching experience with Git so, tell them about how you have used branching in your previous job and what purpose does it serves, you can refer the below mention points:

- Feature branching
  A feature branch model keeps all of the changes for a particular feature inside of a branch. When the feature is fully tested and validated by automated tests, the branch is then merged into master.
- Task branching
  In this model each task is implemented on its own branch with the task key included in the branch name. It is easy to see which code implements which task, just look for the task key in the branch name.
- Release branching
  Once the develop branch has acquired enough features for a release, you can clone that branch to form a Release branch. Creating this branch starts the next release cycle, so no new features can be added after this point, only bug fixes, documentation generation, and other release-oriented tasks should go in this branch. Once it is ready to ship, the release gets merged into master and tagged with a version number. In addition, it should be merged back into develop branch, which may have progressed since the release was initiated.

In the end tell them that branching strategies varies from one organization to another so I know basic branching operations like delete, merge, checking out a branch etc..

**Q19. How will you know in Git if a branch has already been merged into master?**

The answer is pretty direct.

To know if a branch has been merged into master or not you can use the below commands:

**git branch –merged** It lists the branches that have been merged into the current branch.
**git branch –no-merged** It lists the branches that have not been merged.

**Q20. What is Git rebase and how can it be used to resolve conflicts in a feature branch before merge?**

According to me you should start by saying git rebase is a command which will merge another branch into the branch where you are currently working, and move all of the local commits that are ahead of the rebased branch to the top of the history on that branch.

Now, once you have defined Git rebase time for an example to show how it can be used to resolve conflicts in a feature branch before merge

conflicts in a feature branch before merge.

If a feature branch was created from the master, and since then the master branch has received new commits, Git rebase can be used to move the feature branch to the tip of master. The command effectively will replay the changes made in the feature branch at the tip of master, allowing conflicts to be resolved in the process. When done with care, this will allow the feature branch to be merged into master with relative ease and sometimes as a simple fast-forward operation.

*You can also expect some off track questions, so the next question in this Git interview questions blog will be regarding SubGit.*

**Q21. What is SubGit?**

Begin this answer by explaining what is SubGit used for.

SubGit is a tool for SVN to Git migration. It creates a writable Git mirror of a local or remote Subversion repository and uses both Subversion and Git as long as you like.

Now you can include some advantages like you can do a fast one-time import from Subversion to Git or use SubGit within Atlassian Bitbucket Server.We can use SubGit to create a bi-directional Git-SVN mirror of existing Subversion repository. You can push to Git or commit to Subversion at your convenience. Synchronization will be done by SubGit.



I have included the frequently asked Git interview questions. If you have more questions in your mind just type it in the comment box below and we will reply you ASAP. Before going for the interview I will suggest you to ***check out this Git blog series*** (https://www.edureka.co/blog/what-is-git/).

*If you found this **Git Interview Questions** relevant, check out the **DevOps training*** (https://www.edureka.co/devops/) *by Edureka, a trusted online learning company with a network of more than 250,000 satisfied learners spread across the globe. The Edureka DevOps Certification Training course helps learners gain expertise in various DevOps processes and tools such as Puppet, Jenkins, Nagios and GIT for automating multiple steps in SDLC.*

**About Saurabh (22 Posts (https://www.edureka.co/blog/author/saurabhedureka-co/))**

Saurabh is a technology enthusiast working as a Research Analyst at Edureka. His areas of interest are - DevOps, Artificial Intelligence, Big Data and Data Science. Writing about new technologies is his favorite pastime.