**GIT Interview Questions**

**How do you revert a commit that had already been pushed and made public?**

creates a new commit with patches that cancel out the changes introduced in specific commits. In case the commit that needs to be reverted has already been published or changing the repository history is not an option, *git revert* can be used to revert commits. Running the following command will revert the last two commits:

git revert HEAD~2..HEAD

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

Squashing multiple commits into a single commit will overwrite history, and should be done with caution. However, this is useful when working in feature branches. To squash the last N commits of the current branch, run the following command (with {N} replaced with the number of commits that you want to squash):

git rebase -i HEAD~{N}

Upon running this command, an editor will open with a list of these N commit messages, one per line. Each of these lines will begin with the word “pick”. Replacing “pick” with “squash” or “s” will tell Git to combine the commit with the commit before it. To combine all N commits into one, set every commit in the list to be squash except the first one. Upon exiting the editor, and if no conflict arises, *git rebase* will allow you to create a new commit message for the new combined commit.

**What is GIT stash?**

GIT stash takes the current state of the working directory and index and puts in on the stack for later and gives you back a clean working directory.  So in case if you are in the middle of something and need to jump over to the other job, and at the same time you don’t want to lose your current edits then you can use GIT stash.

**What is the function of ‘git stash apply’?**

When you want to continue working where you have left your work, ‘git stash apply’ command is used to bring back the saved changes onto the working directory.

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

Git branch —merged lists the branches that have been merged into the current branch

Git branch —no merged lists the branches that have not been merged

**What is a ‘conflict’ in git?**

A ‘conflict’ arises when the commit that has to be merged has some change in one place, and the current commit also has a change at the same place. Git will not be able to predict which change should take precedence.

**How can conflict in git resolved?**

To resolve the conflict in git, edit the files to fix the conflicting changes and then add the resolved files by running “git add” after that to commit the repaired merge,  run “git commit”.  Git remembers that you are in the middle of a merger, so it sets the parents of the commit correctly.

**What is a Jenkins job?**

A Job/Project is the fundamental unit of a logical work (like a software build, an automation task, test execution, etc) using the Jenkins automation server and other required plugins, configurations & infrastructures.

Jobs can be of different types like - a freestyle project, a multi-configuration project, a pipeline project, a multi-branch project, etc.

**How can you set up a Jenkins job?**

To set up a Jenkins job, you may follow these steps:

* Select New item from the menu
* Next, enter a name for the job and select a free-style job
* Click on OK to create a new job
* Hence, the next page that appears will allow you to configure your job.

**Name some more continuous Integration tools other than Jenkins.**

Some of the top continuous integration tools other than Jenkins are:

* TeamCity
* Travis CI
* Go CD
* Bamboo
* GitLab CI
* CircleCI
* Codeship

**Explain the process in which Jenkins works?**

The process in which Jenkins works is as follows:

* Jenkins checks changes in repositories regularly, and developers must secure their code regularly.
* Once the changes are defined, Jenkins detects them and uses them to prepare a new build.
* After that, Jenkins will transverse through various stages in its usual pipeline. As one stage completes, the process will move further on to the next stage.
* If a stage fails, Jenkins build will stop there, and the software will email the team using it. When completed successfully, the code implements itself in the proper server so that testing begins.
* After the successful testing phase, Jenkins shares the results with the team using it.

**Name the Jenkins suite’s essential plugins?**

The Jenkin suite’s essential plugins are Docker, Jira, Slack Notification, Maven, Amazon E2C, jUnit, Pipeline, Mailer, and Greenballs.

**What is a trigger? Give an example of how the repository is polled when a new commit is detected.**

Triggers are used to define when and how pipelines should be executed.

When Jenkins is integrated with an SCM tool, for example, Git, the repository can be polled every time there is a commit.

* The Git plugin should be first installed and set up.
* After this, you can build a trigger that specifies when a new build should be started. For example, you can create a job that polls the repository and triggers a build when a change is committed.

**Name a Jenkins environment variable you have used in a shell script or batch file.**

There are numerous environment variables that are available by default in any Jenkins build job. A few commonly used ones include:

* $JOB\_NAME
* $NODE\_NAME
* $WORKSPACE
* $BUILD\_URL
* $JOB\_URL

Note that, as new Jenkins plug-ins are configured, more environment variables become available. For example, when the Jenkins Git plug-in is configured, new Jenkins Git environment variables, such as $GIT\_COMMIT and $GIT\_URL, become available to be used in scripts.

**Interview Questions: GIT**

**Q5. What is Git?**

I will suggest that you attempt this question by first explaining about the architecture of git as shown in the below diagram. You can refer to the explanation given below:

* 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 below with “Local repository” and has the full history of the project on his hard drive so that 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”.

A picture containing text, businesscard, screenshot

Description automatically generated

**Q6. Explain some basic Git commands?**

Below are some basic Git commands:

Table

Description automatically generated

**Q7. In Git how do you revert a commit that has already been pushed and made public?**

There can be two answers to this question so 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>**

**Q8. 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})”**

**Q9. 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 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.

**Q10. 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, if a feature branch was created from 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.

**Q11. 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.

**Q12. 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 so you can say that, To get a list files that has changed in a particular commit use 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 mention 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.

**Q13. How do you setup a script to run every time a repository receives new commits through push?**

There are three ways to configure a script to run every time a repository receives new commits through push, one needs to define either a pre-receive, update, or a post-receive hook depending on when exactly the script needs to be triggered.

* Pre-receive hook in the destination repository is invoked when commits are pushed to it. Any script bound to this hook will be executed before any references are updated. This is a useful hook to run scripts that help enforce development policies.
* Update hook works in a similar manner to pre-receive hook, and is also triggered before any updates are actually made. However, the update hook is called once for every commit that has been pushed to the destination repository.
* Finally, post-receive hook in the repository is invoked after the updates have been accepted into the destination repository. This is an ideal place to configure simple deployment scripts, invoke some continuous integration systems, dispatch notification emails to repository maintainers, etc.

Hooks are local to every Git repository and are not versioned. Scripts can either be created within the hooks directory inside the “.git” directory, or they can be created elsewhere and links to those scripts can be placed within the directory.

**Q14. How will you know in Git if a branch has already been merged into master?**

I will suggest you to include both the below mentioned commands: git branch –merged lists the branches that have been merged into the current branch. git branch –no-merged lists the branches that have not been merged.