Advanced GIT Interview Questions

1. How will you resolve conflict in Git?

- Conflicts occur whenever there are multiple people working on the same file across multiple branches. In such cases, git won't be able to resolve it automatically as it is not capable of deciding what changes has to get the precedence.
- Following are the steps are done in order to resolve git conflicts:
  1. Identify the files that have conflicts.
  2. Discuss with members who have worked on the file and ensure that the required changes are done in the file.
  3. Add these files to the staged section by using the git add command.
  4. Commit these changes using the git commit command.
  5. Finally, push the changes to the branch using the git.

2. What command helps us know the list of branches merged to master?

- git branch --merged helps to get the list of the branches that have been merged into the current branch.
- Note: git branch --no-merged lists the branches that have not been merged to the current branch.

3. What is best advisable step in cases of broken commit: Create an additional commit OR amend an existing commit?

- It is always advisable to create an additional commit rather than amending the existing commit due to the following reasons:
  - Doing the amend operation destroys the previously saved state of that commit. If only the commit message gets changes or destroyed, it's acceptable but there might be cases when the contents of the commits get amended. This results in the loss of important information associated with the commit.
  - Over usage of git commit --amend can have severe repercussions as the small commit amend can continue to grow and gather unrelated changes over time.

4. How to revert a bad commit which is already pushed?

There can be cases where we want to revert from the pushed changes and go back to the previous version. To handle this, there are two possible approaches based on the situations:

- **Approach 1**: Fix the bad changes of the files and create a new commit and push to the remote repository. This step is the simplest and most recommended approach to fix bad changes. You can use the command: git commit -m "<message>"
- **Approach 2**: New commit can be created that reverts changes done in the bad commit. It can be done using git revert <name of bad commit>

5. What is the functionality of "git cherry-pick" command?

This command is used to introduce certain commits from one branch onto another branch within the repository. The most common use case is when we want to forward- or back-port commits from the maintenance branch to the development branch.

6. Explain steps involved in removing a file from git index without removing from the local file system?

- Sometimes we end up having certain files that are not needed in the git index when we are not being careful while using the git add command. Using the command git rm will remove the file from both the index and the local working tree which is not always desirable.
- Instead of using the git rm command we can use the git reset command for removing the file from the staged version and then adding that file to the .gitignore file to avoid repeating the same mistake again.

git reset <file_name> # remove file from index

echo filename >> .gitingore # add file to .gitignore to avoid mistake repetition.

7. What are the factors involved in considering which command to choose among: git merge and git rebase?

Both these commands ensure that changes from one branch are integrated into another branch but in very different ways. Git rebasing can be thought of as saying to use another branch as a new base for the work.

- Whenever in doubt, it is always preferred to use the git merge command.

  Following are some factors that tell when to use merge and rebase commands:

- In case our branch gets contributions from other developers outside the team as in open-source or public repositories, then rebase is not preferred.
  - This is because rebase destroys the branch and it results in broken and inconsistent repositories unless the git pull --rebase command is used.
- Rebase is a very destructive operation. If not applied correctly, it results in loss of committed work which might result in breaking the consistency of other developer's contribution to the repository.
- If the model of having branches per feature is followed, rebasing is not a good idea there because it keeps track of related commits done by the developers. But in case the team follows having branches per developer of the team, then the branch has no additional useful information to be conveyed. In this model, rebasing has no harm and can be used.
- If there is any chance where there might be a necessity to revert a commit to previous commits, then reverting a rebase would be almost impossible as the commit data would be destroyed. In such cases, the merge can be used.

8. How do you find a commit which broke something after a merge operation?

- This can be a time-consuming process if we are not sure what to look at exactly. Fortunately, git provides a great search facility that works on the principle of binary search as git-bisect command.
- The initial set up is as follows:

```
git bisect start        # initiates bisecting session
git bisect bad          # marks current revision as bad
git bisect good revision # marks last known commit as good revision
```

- Upon running the above commands, git checks out a revision that is labeled as halfway between "good" and "bad" versions. This step can be run again by marking the commit as "good" or "bad" and the process continues until the commit which has a bug is found.

9. What are the functionalities of git reset --mixed and git merge --abort?

- git reset --mixed command is used for undoing changes of the working directory and the git index.
- git merge --abort command is used for stopping the merge process and returning back to the state before the merging occurred.

10. Can you tell the differences between git revert and git reset?

git revert	git reset
This command is used for creating a new commit that undoes the changes of the previous commit.	This command is used for undoing the local changes done in the git repository
Using this command adds a new history to the project without modifying the existing history	This command operates on the commit history, git index, and the working directory.