**Git Interview Questions and Answers  
  
  
1. What is a Git repository?**  
A Git repository stores a project's files and revision history and facilitates version control by tracking changes made over time. It can be located locally within a folder on your device or an online platform like GitHub. This enables users to collaborate, revert to previous versions, and efficiently manage project development using commands like commit, push, and pull.

**2. How does Git work?**  
Git operates by recording changes made to files and directories in a project, capturing snapshots of its evolving condition. Users can oversee alterations, create branches for simultaneous development, merge branches, and revert to previous states if required. It also promotes collaboration and ensures effective version control in software development endeavors.

**3. What is git add?**  
The git add command is used in Git to stage changes for inclusion in the next commit. It prepares modifications, additions, or deletions made to files in the working directory, marking them to be included in the upcoming commit snapshot. Note this command does not actually commit the changes but prepares them for staging.

**4. What is git push?**  
The git push command is used in Git to upload local repository content to a remote repository. It transfers committed changes from the local repository to a remote one, typically on a server like GitHub or GitLab. This command enables collaboration by allowing users to share their changes with others on the same project.

**5. What is git status?**  
The git status command displays the current state of the repository in Git. It provides information about which files have been modified, which are staged for the next commit, and which are untracked. It helps users track the progress of their work and identify any changes that need to be committed or staged.

**6. What is a commit in Git?**  
A commit represents a snapshot of the changes made to files in a repository at a specific point in time. When you commit changes in Git, you are effectively saving the current state of your files and can provide a descriptive message that explains the changes made (which is recommended).

Each commit creates a unique identifier, allowing you to track the history of changes in the repository. Commits play a crucial role in version control, as they provide a way to revert to previous states of the project, review the history of changes, and collaborate with others by sharing updates.

**7. What is branching in Git?**  
Branching refers to the practice of diverging from the main line of development (typically called the "master" branch) to work on new features, fixes, or experiments without affecting the main codebase. It allows multiple parallel lines of development to coexist within the same repository.

Each branch represents a separate line of development with its own set of commits, enabling developers to work on different features or fixes simultaneously. Branching facilitates collaboration, experimentation, and organization within a project, as changes made in one branch can be merged back into the main codebase once they are completed and tested.

**8. What is a conflict in Git?**  
Conflicts arise when conflicting changes are made to the same part of a file or files by different contributors, typically during a merge or rebase operation. Git cannot automatically resolve these conflicting changes, requiring manual intervention by the user to resolve the discrepancies.

Thus, to resolve conflicts, the conflicted files must be reviewed and edited based on the best suited reconciliation before the resolved version is committed.

**9. What is merge in Git?**  
Merging is a fundamental operation in Git that facilitates collaboration and the integration of changes across different branches in a project. Namely, a merge is the process of combining the changes from different branches into a single branch, typically the main branch (e.g., master or main).

A merge integrates the changes made in one branch with another, resulting in a new commit that combines the histories of both branches.

**10. What is a remote in Git?**  
A remote is a repository hosted on a server or another computer for collaboration and sharing code with others. It serves as a centralized location where developers can push their local changes and pull changes made by others.

Remotes are typically set up on hosting platforms like GitHub, GitLab, or Bitbucket, and they enable distributed development and facilitate teamwork by providing a common location for storing and synchronizing project code among multiple contributors.

**11. What is the difference between git fetch and git pull?**  
The main difference between git fetch and git pull lies in what they do and how they update the local repository.

* **git fetch** retrieves changes from a remote repository to the local repository. It updates the remote-tracking branches (e.g., origin/master) but does not update the working directory or merge any changes into the current branch.
* **git pull** retrieves changes and merges them into the current branch in one step. It essentially performs a git fetch followed by a git merge.

**12. How do you revert a commit that has already been pushed and made public?**  
The git revert command can be used to revert a commit that has already been pushed and made public.

1. Identify the commit hash using the git log command.
2. Run git revert <commit-hash> to create a new commit that undoes the changes.
3. Edit the commit message if needed and save it.
4. Push the new commit using git push origin <branch-name>.

Using git revert is safer than git reset as it does not alter the commit history.

**13. What does git reset do?**  
The git reset command resets the current HEAD to a specified state. Modes of reset:

* **--soft**: Resets the HEAD pointer, keeping changes staged.
* **--mixed**: Unstages changes but keeps them in the working directory.
* **--hard**: Discards all changes.

**14. What is git stash?**  
git stash temporarily stores changes in the working directory that are not ready to be committed. It saves modifications without committing them and allows you to reapply them later.

**15. What is git reflog?**  
git reflog records changes to the HEAD pointer and provides a chronological list of recent actions. It helps recover lost commits or branches.

**16. How do you make an existing Git branch track a remote branch?**  
Use the following command:

git branch --set-upstream-to=<remote-name>/<branch-name>  
Or:  
git branch -u <remote-name>/<branch-name>

**17. How do you manage multiple configurations for different projects in Git?**  
Utilize git config with --local, --global, or --system flags, or use includeIf to include specific configurations based on the repository path.

**18. How do you handle large files with Git?**  
Use **Git LFS** (Large File Storage) to store large files outside the repository while keeping lightweight pointers.

**19. What is the use of git submodule and how do you update one?**  
A submodule allows inclusion of external repositories within a main repository. To update:

1. Navigate to the submodule directory.
2. Run git fetch or git pull.
3. Commit the updated submodule to the main repository.

**20. What is the significance of git push --force-with-lease over git push --force?**  
git push --force-with-lease is safer than git push --force as it prevents overwriting changes made by others. It rejects the push if the remote branch has updates unknown to you.

**Basic Git Interview Questions for Freshers**

**1. What is Git?**

Git is a distributed version control system (DVCS) that is used to track changes in source code during software development. It permits multiple developers to work on a project together without interrupting each other's changes. Git is especially popular for its speed, and ability to manage both small and large projects capably.

**2. What is a repository in Git?**

A Git repository (or repo) is like a file structure that stores all the files for a project. It continues track changes made to these files over time, helping teams work together evenly. Git can control both local repositories (on your own machine) and remote repositories (usually hosted on platforms like GitHub, GitLab, or Bitbucket), allowing teamwork and backup.

**3.**[**What is the difference between Git and GitHub**](https://www.geeksforgeeks.org/difference-between-git-and-github/)**?**

| **Git** | **GitHub** |
| --- | --- |
| Git is a version control system used to track changes in files over time | GitHub is a platform where Git repositories can be stored and shared |
| It runs locally on your computer | It is a cloud-based service |
| Git can be used offline, as it operates locally on your machine. | GitHub requires an internet connection because it is hosted on the web |

**4.**[**What is origin in Git**](https://www.geeksforgeeks.org/git-origin-master/)**?**

In Git, "origin" states to the default name offered to the remote repository from which local repository was cloned. It is used as a reference to control fetches, pulls, and pushes.

**5. What is the purpose of the**[**.gitignore file?**](https://www.geeksforgeeks.org/what-is-git-ignore-and-how-to-use-it/)

The '.gitignore' file tells Git which files and folders to ignore when tracking changes. It is used to avoid attaching unneeded files (like logs, temporary files, or compiled code) to your repository. This saves repository clean and targeted on important files only.

**6. What is a version control system (VCS)?**

A [version control system](https://www.geeksforgeeks.org/version-control-systems/) (VCS) records the work of developers coordinating on projects. It keeps the history of code changes, permitting developers to add new code, fix bugs, and run tests securely. If required, they can restore a past working version, verifying project security.

**7. What is the git push command?**

The '[git push](https://www.geeksforgeeks.org/what-is-git-push/)' command is used to share local repository changes to a remote repository. It changes the remote repository with the recent commits from the fixed local branch.

**8. What is the git pull command?**

The '[git pull](https://www.geeksforgeeks.org/what-is-git-pull/)' command updates the current local branch with changes from a remote repository and combining it with a local repository.

**9. What does git clone do?**

The 'git clone' forms a copy of a remote repository upon your local machine. It downloads all files, branches, and history, enabling you to start working on the project or contribute to it. With git clone -b , you can download and work on an individual branch of a repository.

**10. What are the advantages of using GIT?**

Using Git provides multiple advantages:

* It assists teamwork by supporting multiple developers to work on the same project together.
* Each developer has a local copy of the repository, improving performance and enabling offline work.
* Free and widely supported.
* Git supports work on various types of projects.
* Each repository has only one Git directory.

**11. What is the difference between git init and git clone?**

'git init' develops a new, empty Git repository in the present directory, while 'git clone' copies an existing remote repository, containing all files and history, to a local directory.

**12. What is git add?**

The '[git add](https://www.geeksforgeeks.org/what-is-git-add/)' command marks changes in your project for the next commit. It tells Git which files to involve in the later update, making them ready to be saved in the repository. This is the early step in recording changes in the Git repository.

**13. What is git status?**

The 'git status' command shows the recent status of your Git repository. It tells you which files have changed, which ones are ready to be committed, and which ones are new and unobserved. This benefits you monitor your work's growth and see what changes want to be set up or committed.

**14. What is a commit in Git?**

A [commit](https://www.geeksforgeeks.org/what-is-git-commit/) in Git denotes a snapshot of changes made to files in a repository. It grabs all the changes you have made to files—like additions, or deletions of files at a particular moment. Each commit has a unique message explaining what was done. This helps you track your project's history, undo changes if requisite, and work with others on the same project.

**15. What is the purpose of the git clean command?**

The 'git clean' command is used to erase ignored files from the working directory of Git repository. Its motive is to clean up the workspace by deleting files that are not being saved by Git, checking a clean state with only observed files present.

**16. What is a ‘conflict’ in git?**

Git usually manages merges automatically, but conflicts occur when two branches edit the same line or when one branch deletes a file that another edits.

**17. What is the meaning of “Index” in GIT?**

In Git, the "Index" (also called as the "Staging Area") is a place where alterations are temporarily store before committing them to the repository. It permits you to select and prepare specific alterations from your working directory before properly saving them as part of the project's history.

**18. How do you change the last commit in git?**

To change the preceding commit in Git, use 'git commit --amend' after making changes, stage them with 'git add' , and save with the editor.

**19. What is `git checkout`?**

'[git checkout](https://www.geeksforgeeks.org/git-checkout-and-merge/)' helps you switch between branches or return files to a previous state in Git. Now, it is suggested to use 'git switch' for changing branches and 'git restore' to return files. These commands are more intent on their particular tasks for better clearness and capability.

**20. How do you switch branches in Git?**

To switch branches in Git, use 'git checkout ' to move to a present branch. On the other hand, use git switch in newer Git versions for the same objective. This permits you to work on different versions or features of your project stored in separate branches.

**21. Name some popular Git hosting services?**

* [GitHub](https://www.geeksforgeeks.org/introduction-to-github/)
* [GitLab](https://www.geeksforgeeks.org/gitlab/)
* SourceForge.net
* [Bitbucket](https://www.geeksforgeeks.org/introduction-to-bitbucket/)
* Visual Studio Online

**Intermediate** **Git Interview Questions and Answers**

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

'git fetch' fetches updates from a remote repository but does not combine them into your local repository. It fetches all the new data from the remote repository that you don’t have yet, but it stores it in a separate area, permitting you to review the changes before merging them into your working directory.

'[git pull](https://www.geeksforgeeks.org/what-is-git-pull/)' fetches the updates from the remote repository and instantly strives to merge them into your current branch. It is basically a union of 'git fetch' followed by 'git merge' .

**23. Explain Git rebase and when do you use it?**

Git rebase is a process to combine alterations from one branch into another. It forms a linear history, avoiding merge commits. Use it to clean up commit history, keep a project history sequential, and make feature branches up-to-date before uniting.

**24. How will you create a git repository?**

* Download Git on your system if you have not already.
* Create a project folder in the location where you want your repository.
* Open Terminal or Command Prompt and guide to your project folder.
* Run 'git init' in the project folder. This will create a '.git' folder, showing your repository is set.

**25. What differentiates between the command’s git remote and git clone?**

**'git clone'**: Downloads a full copy of a remote repository to your local computer, involving all files and history.

**'git remote'** : Controls connections to remote repositories. It sets up links to remote repositories but doesn't download any files.

**26. What are the benefits of using a pull request in a project?**

Teams can together work on distinct parts of the system and later combine their changes using pull requests. This way boosts team capability.

**27. What is a Git bundle?**

A Git bundle is a collective file that wraps all data from Git repository, such as commits, branches, and tags. It acts as a handy approach for relocating a repository offline or sharing upgrades when network connection is not available. To form a git bundle, perform the following command:

**git bundle create <bundle\_file> <refs>**

**28. What are the advantages of Git over SVN?**

* Git permits developers to work offline and autonomously, However SVN needs a steady network connection to commit changes.
* Git is broadly rapid for most operations, containing branching, merging, and committing, due to its distributed nature and efficient data handling.
* Git supports various development workflows, such as feature branching, pull requests, and decentralized collaboration, which is adaptable to different team sizes and project needs.

**29. What is git stash?**

Git stash is Git command used to temporarily store changes in your working directory that are not yet ready to be committed. It permits developers to conserve modifications without committing them to the repository.

**30. How do you revert a commit that has already been pushed and made public?**

To revert a commit that has been pushed and made public, follow these steps:

* Checkout the Branch: Switch to the branch where you want to revert the commit.

git checkout <branch-name>

* Find the Commit to Revert: Use 'git log' to find the commit hash of the commit you want to revert.

git log

* Revert the Commit: Use 'git revert' followed by the commit hash of the commit you want to revert.

git revert <commit-hash>

* Review Changes: Git will open your default text editor to confirm the revert message. Save and close the editor to proceed.
* Push the Revert: Finally, push the reverted commit to the remote repository.

git push origin <branch-name>

**31. Explain the difference between reverting and resetting?**

**Resetting:** Resetting: This command is used to change the present state of the repository to a precise point in its history. When you refresh, Git moves the 'HEAD' (present branch) to the particular commit, likely changing the files in your working directory and staging area. It is like reversing to a definite point in time, and it can be used to discard changes.

**Reverting:** Reverting, on the other a hand, makes new commit that undoes the changes made by specific commit. In place of removing or changing history like resetting does, reverting adds new commit that effectively reverses the changes introduced by the commit you specify.

**32. What is the difference between git reflog and log?**

**Git Log:** It shows a history of commits (changes) made in your Git repository. It lists out commits in linear order, showing who made each change, when they made it, and the commit message.

**Git Reflog:** It stands for "reference log". It records changes to the HEAD (current branch pointer) and permits you to see a timeline of recent movements within the repository, like commits, checkouts, merges, etc. It is effective for recovering lost commits or branches that are no longer visible in the regular log.

**33. What is the HEAD in Git?**

In Git, HEAD is a source to the current branch or commit you are working on. HEAD normally shows the recent commit of the current branch and moves when you switch branches or check out exact commits.

**34. What is the purpose of `git tag -a`?**

The intent of 'git tag -a' is to form an annotated tag in Git. Annotated tags are tags that contain additional metadata such as the tagger's name, email, date, and a message. They are valuable for labeling important points in history, like releases, and give another context compared to lightweight tags made with 'git tag' .

**35. What is the difference between `HEAD`, `working tree` and `index` in Git?**

**HEAD:** Points to the currently checked-out commit or branch.

**Working Tree:** Includes the actual files you are currently working on or modifying.

**Index (Staging Area):** Acts as a middle ground where changes are prepared before committing. It keeps snapshots of change to be included in the next commit.

**36. How to resolve a conflict in Git?**

**Discover Conflicting Files:** Find the files where conflicts have take place because of changes from unique branches.

**Correct Files to Repair Conflicts:** Alter the files to sync conflicting changes and block future conflicts.

**Stage solved Files:** Use 'git add' to add the resolved files to the staging area. Commit the Changes: Complete the process by committing the changed files applying 'git commit' .

**37. Explain the difference between `git merge` and `git rebase` and when you would use each?**

'[git merge](https://www.geeksforgeeks.org/git-merge/)' merges changes from one branch into another, keeping distinguish branch histories. It forms merge commits that directly show where branches came together. '[git rebase](https://www.geeksforgeeks.org/how-to-use-git-rebase/)' reforms history by using commits from one branch onto another, causing in a linear sequence of commits. Prefer 'merge' to keep distinct branch timelines and use 'rebase' for enhancing commit history before merging branches.

**38. What language is used in GIT?**

Git is mainly developed using the [C programming language](https://www.geeksforgeeks.org/c-programming-language/). The core features and commands of Git, containing its data structures and algorithms, are applied in C. This choice of language confirms productivity, speed, and portability across distinct operating systems and platforms.

**39. How do you add a file to the staging area?**

Use `git add <file\_name>` to add a file to the staging area, forming it ready for a commit.

**40. What is `git diff`?**

'git diff' is a command in Git that presents the differences between varied states of files in a repository. It equates changes between the working directory, the staging area (index), and the last commit. It assists track changes, additions, and deletions before committing changes to the repository.

**Advanced Git Interview Questions for Experienced**

**41. What is the Git object model?**

The Git object model comprises four major types: blobs (which store file data), trees (which store directory structures), commits (which store repository snapshots), and tags (which store references to commits). These objects are the pillar of Git's version control system, permitting for capable tracking and management of changes.

**42. Explain `git rebase` and when you would use each?**

'git rebase' moves commits from one branch to another, making a straight, linear history. Use rebase to enhance and clean up the commit history before merging.

**43. What is a git hook and how might you use it?**

A Git hook is a pattern script that instantly runs at exact points in the Git workflow, such as before or after commits, merges, or pushes. You can use hooks to apply coding rules, run tests, check for security faults, or simplify tasks. For an example, a pre-commit hook can run and tests to verify code quality before changes are committed.

**44. How does Git store data?**

Git stores data by saving snapshots of your project at diverse points in time. Each snapshot is a commit, which covers information about the project’s files (blobs) and directories (trees). These snapshots are recognized by unique hashes, creating it easy to track changes and retrieve history.

**45. Explain what is meant by the "detached HEAD" state in Git?**

When 'HEAD' points directly to a commit instead of a branch, Git is in a detached 'HEAD' state. Changes made in this state won’t be referenced by any branch and might be lost if not saved.

**46. Explain the difference between git reset, git revert, and git checkout?**

* **git reset:** Moves HEAD to different commit, potentially changing history.
* **git revert:** Undoes exact commit by making new commit with inverse changes.
* **git checkout:** Switches branches or checks out files from commit, putting you in "detached HEAD" state for direct commits.

**47. How do you handle large files with Git?**

To handle large files in Git, use Git [LFS](https://www.geeksforgeeks.org/git-lfs-large-file-storage/) (Large File Storage). It tracks large files severally from your repository, storing them on a remote server. This prevents bloating your repository size and secures improved performance while operations like cloning and fetching.

**48. What is the function of the git cherry-pick command?**

The git [cherry-pick command](https://www.geeksforgeeks.org/how-to-use-the-cherry-pick-command-in-git/) uses exact commits from one branch to another, allowing selective merging of changes without merging entire branches.

**49. What is ‘bare repository’ in Git?**

A [bare repository](https://www.geeksforgeeks.org/bare-repositories-in-git/) in Git is one missing a working directory. It only contains version control data, making it ideal for sharing and collaboration without changing files directly.

**50. What is branching in Git?**

[Branching](https://www.geeksforgeeks.org/branching-strategies-in-git/) in Git permits forming separate lines of development. It allows users to work on features or fixes separately from the main codebase, helping parallel development and simpler integration of changes.

**Git Interview Questions and Answers**

**1. Explain the difference between rebasing and merge in Git?**

• Git rebase is a command that allows developers to integrate changes from one branch to another.  
• Git merge is a command that allows you to merge branches from Git.

Git rebase and merge both integrate changes from one branch into another. Where they differ is how they used. Git rebase moves a feature branch into a master. Git merge adds a new commit, preserving the history.

(If you’re working alone or on a small team, use rebase. If you’re working with a big team, use merge.)

**2. Have you faced the situation where you resolve conflicts in Git? How?**

A merge conflict is an event that takes place when Git is unable to automatically resolve differences in code between two commits. Git can merge the changes automatically only if the commits are on different lines or branches. Here are the steps that will help you resolve conflicts in Git:  
1. The easiest way to resolve a conflicted file is to open it and make any necessary changes  
2. After editing the file, we can use the git add a command to stage the new merged content  
3. The final step is to create a new commit with the help of the git commit command  
4. Git will create a new merge commit to finalize the merge

**3. How to revert a commit that has already been pushed and made public?**

There are two processes through which you can revert a commit:  
1. Remove or fix the bad file in a new commit and push it to the remote repository. Then commit it to the remote repository using:  
git commit –m “commit message”  
2. Create a new commit to undo all the changes that were made in the bad commit. Use the following command:  
git revert <commit id>

**4. Tell about the commands git reset — mixed and git merge — abort?.**

**git reset**— mixed is used to undo changes made in the working directory and staging area.  
**git merge**— abort helps stop the merge process and return back to the state before the merging began.

**5. How will you find a list of files that has been modified in a particular commit?**

The command to get a list of files that has been changed in a particular commit is:  
git diff-tree –r {commit hash}  
• -r flag allows the command to list individual files  
• commit hash lists all the files that were changed or added in the commit.

**6. How will you fix a broken commit? What command you will use?**

To fix a broken commit in Git, We use the “git commit — amend” command, which helps us combine the staged changes with the previous commits instead of creating a fresh new commit.

**7. Explain git stash drop?**

Git ‘stash drop’ command is used to remove the stashed item. This command will remove the last added stash item by default, and it can also remove a selected item as well.  
Ex: If you want to delete item named stash@{manoj}; you can use the command:  
*git stash drop stash@{manoj}.*

**8. Explain about “git cherry-pick”?**

This command enables you to pick up commits from a branch within a repository and apply it to another branch. This command is useful to undo changes when any commit is accidentally made to the wrong branch. Then, you can switch to the correct branch and use this command to git cherry-pick the commit.

**9. Can you tell 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 pull = git fetch + git merge)

**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.

**10. What is origin in Git?**

Origin refers to the remote repository that a project was originally cloned from and is used instead of the original repository’s URL.

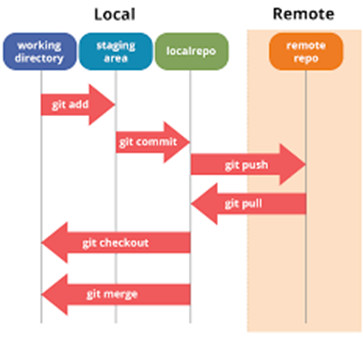
**11. What is the difference between resetting and reverting?**

git reset changes the state of the branch to a previous one by removing all of the states after the desired commit,

git revert does it through the creation of new reverting commits and keeping the original one intact.

**12. What is ‘staging area’ or ‘index’ in Git?**

That before completing the commits, it can be formatted and reviewed in an intermediate area known as ‘Staging Area’ or ‘Index’. Every change is first verified in the staging area and then that change is committed to the repository.

  
  
**13. What work is restored when the deleted branch is recovered?**

The files which were stashed and saved in the stash index list will be recovered back. Any untracked files will be lost. Also, it is a good idea to always stage and commit your work or stash them.

**14. What is Head in Git?**

Git maintains a variable for referencing, called HEAD to the latest commit in the recent checkout branch. So if we make a new commit in the repo then the pointer or HEAD is going to move or change its position to point to a new commit.

**15. What is the purpose of branching and its types?**

It allows the user to switch between the branches to keep the current work in sync without disturbing master branches and other developer’s work as per their requirements.

**· 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 branch, 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.

**Basic Git Commands — Refresh your mind once again**

**git init:** creating a new repository.  
**git clone:**to copy or check out the working repository.  
**git pull:**fetch the code already in the repository.  
**git push:** sending the changes to the master branch.  
**git add:** It adds file changes in an existing directory to index.  
**git commit –m [type in a message]** — It is used to snapshot or record a file.  
**git diff [first branch] [second branch]**— it is used to display the differences present between the two branches.  
**git rest [commit]** — It is used to undo all the changes that have been incorporated as a part of a commit after a specified commit has taken place.  
**git reset –hard [commit]** — This command is used to discard all the history and takes us to the last specified commit.  
**git log –follow [file]** — his is similar to that of git log with the additional difference that it lists the version history for a particular file.  
**git show [commit]** — This is used to display the metadata and all the content related changes of a particular commit.  
**git tag [commitID]**— This is used to give particular tags to the code commits.  
**git branch [branch-name]** — This is used to create a new branch.  
**git branch –d [branch name]** — It is used to delete the current branch name specified.  
**git checkout [branch-name]**— It is helpful in switching from one branch to another.  
**git status:** To know the comparison between the working directories and index.

**Basic GIT Interview Questions**

**1. What is a git repository?**

A repository is a file structure where git stores all the project-based files. Git can either stores the files on the local or the remote repository.

**Create a free personalised study plan**

Get into your dream companies with expert guidance

Real-Life Problems

Prep for Target Roles

Custom Plan Duration

**2. What does git clone do?**

The command creates a copy (or clone) of an existing git repository. Generally, it is used to get a copy of the remote repository to the local repository.

**3. What does the command git config do?**

The git config command is a convenient way to set configuration options for defining the behavior of the repository, user information and preferences, git installation-based configurations, and many such things.   
  
For example:  
To set up your name and email address before using git commands, we can run the below commands:

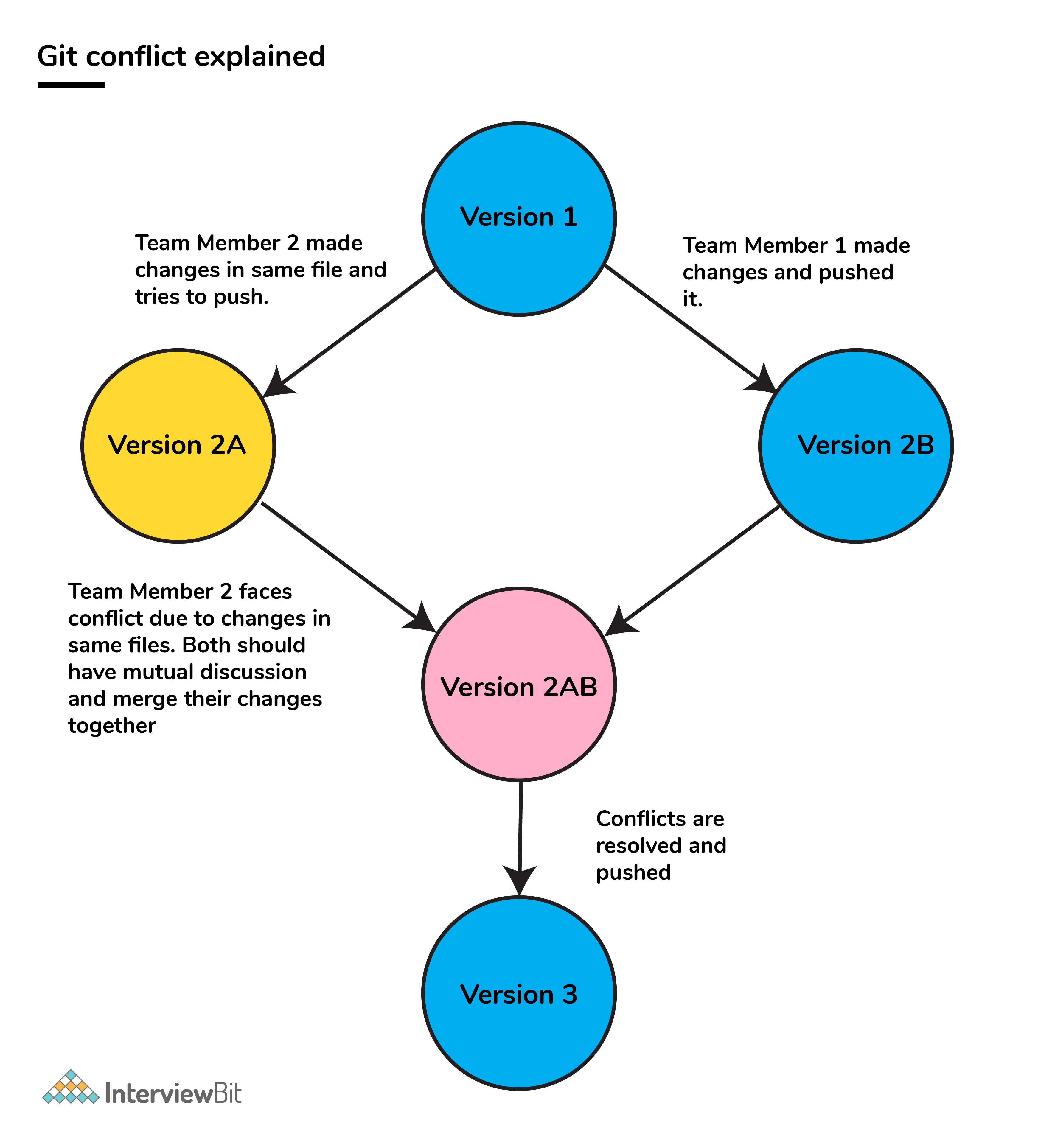
* git config --global  
  user.name  
  “<<your\_name>>”
* git config --global user.email “<<your\_email>>”

**4. Can you explain head in terms of git and also tell the number of heads that can be present in a repository?**

* A head is nothing but a reference to the last commit object of a branch.
* For every repository, there will always be a default head referred to as “master” or now “main” (as per GitHub) but there is no restriction to the count of heads available. In other words, it can have any number of heads.
* **Usages:**  
    
  - To go or checkout to 1 commit before the latest commit, we use git checkout HEAD~1  
    
  - To uncommit the last 3 commits without losing the changes, we first run git reset HEAD~3. Then we can see the changes made in the last 3 commits and then update it manually and commit it finally.  
    
  - In order to uncommit the last 3 commits and also remove the changes, we can run the command: git reset --hard HEAD~3. This command will completely remove all the changes.  
    
  - To look into the changes made in the last 3 commits, we can run git diff HEAD~3  
    
  - To make a new commit by reverting the last 3 commits, we can run the command: git revert --no-commit HEAD~3...HEAD

**5. What is a conflict?**

* Git usually handles feature merges automatically but sometimes while working in a team environment, there might be cases of conflicts such as:  
    
  1. When two separate branches have changes to the same line in a file  
  2. A file is deleted in one branch but has been modified in the other.
* These conflicts have to be solved manually after discussion with the team as git will not be able to predict what and whose changes have to be given precedence.

  
  
**6. What is the functionality of git ls-tree?**

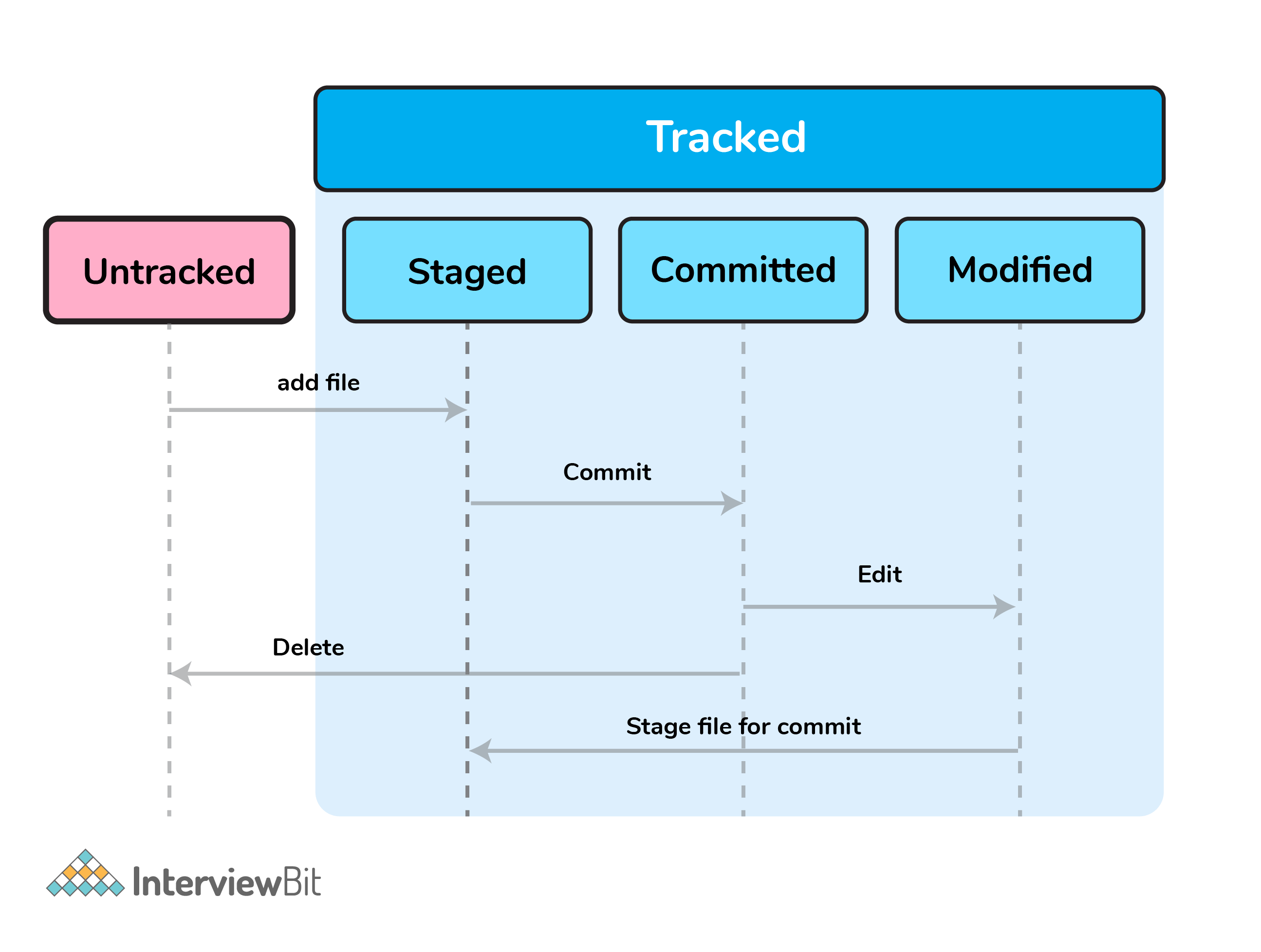
This command returns a tree object representation of the current repository along with the mode and the name of each item and the SHA-1 value of the blob.

**7. What does git status command do?**

git status command is used for showing the difference between the working directory and the index which is helpful for understanding git in-depth and also keep track of the tracked and non-tracked changes.

**8. Define “Index”.**

Before making commits to the changes done, the developer is given provision to format and review the files and make innovations to them. All these are done in the common area which is known as ‘Index’ or ‘Staging Area’.

  
  
In the above image, the “staged” status indicates the staging area and provides an opportunity for the people to evaluate changes before committing them.  
  
**9. What does git add command do?**

* This command adds files and changes to the index of the existing directory.
* You can add all changes at once using git add . command.
* You can add files one by one specifically using git add <file\_name> command.
* You can add contents of a particular folder by using git add /<folder\_name>/ command.

**10. What is a version control system (VCS)?**

A VCS keeps track of the contributions of the developers working as a team on the projects. They maintain the history of code changes done and with project evolution, it gives an upper hand to the developers to introduce new code, fixes bugs, and run tests with confidence that their previously working copy could be restored at any moment in case things go wrong.

**Intermediate GIT Interview Questions**

**1. What has to be run to squash multiple commits (last N) into a single commit?**

Squashing multiple commits to a single one overwrites the history which is why it is recommended to be done using full caution. This step can be done by running the command: git rebase -i HEAD~{{N}} where {{N}} represents the number of commits needed to be squashed.

**2. How would you recover a branch that has already pushed changes in the central repository but has been accidentally deleted from every team member’s local machines?**

We can recover this by checking out the latest commit of this branch in the reflog and then checking it out as a new branch.

**3. Can you tell something about git reflog?**

This command tracks every single change made in the repository references (that can be branches or tags) and also maintains the branches/tags log history that was either created locally or checked out. Reference logs such as the commit snapshot of when the branch was created or cloned, checked-out, renamed, or any commits made on the branch are maintained by Git and listed by the ‘reflog’ command.

* This recovery of the branch is only possible when the branch was either created locally or checked-out from a remote repository in your local repository for Git to store its reference history logs.
* This command should be executed in the repository that had the lost branch.

**4. What consists of a commit object?**

A commit object consists of the following components:

* A set of files that represents the state of a project at a given point in time.
* Reference to parent commit objects.
* A 40 character string termed as SHA-1 name uniquely identifies the commit object.

**5. Explain the levels in git config and how can you configure values using them?**

* In order to make git work, it uses a set of configurations that are pre-defined by default by means of configuration files (or config files). We can change the default behavior of git by just modifying these files which are basically text files. In order to do this, it is important to understand how git identifies these files. It does so by following the below steps:  
    
  - Firstly, git searches for the config values in the system-wide gitconfig file stored in <<installation\_path>>/etc/gitconfig file that has settings defined and applied to **every user** of the system and all their repos.  
       - In case you want git to search from this particular file and read/write on it, we can pass the option --system to git config command.  
    
  - Next, git searches for the ~/.gitconfig file or ~/.config/git/config that has the scope specific to the user.  
     - Git can be made to read/ write from this file specifically bypassing --global to the git config command.  
    
  - Lastly, git searches for the config values in the git directory of the local repository that we are currently working on.  
     - These config values are specific to that particular repository alone and can be accessed by passing --local to the git config command.This is the default config file that gets accessed and modified upon in case we do not specify any levels.

**6. What is a detached HEAD and what causes this and how to avoid this?**

Detached HEAD indicates that the currently checked-out repository is not a local branch. This can be caused by the following scenarios:

* When a branch is a read-only branch and we try to create a commit to that branch, then the commits can be termed as “free-floating” commits not connected to any branch. They would be in a detached state.
* When we checkout a tag or a specific commit and then we try to perform a new commit, then again the commits would not be connected to any branch. When we now try to checkout a branch, these new commits would be automatically placed at the top.  
    
  In order to ensure that detached state doesn't happen, =instead of checking out commit/tag, we can create a branch emanating from that commit and then we can switch to that newly created branch by using the command: git checkout -b <<new\_branch\_name>>. This ensures that a new branch is checkout out and not a commit/tag thereby ensuring that a detached state wouldn't happen.

**7. What does git annotate command do?**

* This command annotates each line within the given file with information from the commit which introduced that change. This command can also optionally annotate from a given revision.
* Syntax: git annotate [<options>] <file> [<revision>]
* You can get to learn more about this command from the official git documentation

**8. What is the difference between git stash apply vs git stash pop command?**

* git stash pop command throws away the specified stash (topmost stash by default) after applying it.
* git stash apply command leaves the stash in the stash list for future reuse. In case we wanted to remove it from the list, we can use the git stash drop command.

git stash pop = git stash apply + git stash drop

**9. What do the git diff and git status commands do?**

| **git diff** | **git status** |
| --- | --- |
| This shows the changes **between commits, working trees,** etc. | This shows the difference **between the working directory and index** that is essential in understanding git in depth. |

git diff works in a similar fashion to git status with the only difference of showing the differences between commits and also between the working directory and index.

**10. Why is it considered to be easy to work on Git?**

With the help of git, developers have gained many advantages in terms of performing the development process faster and in a more efficient manner. Some of the main features of git which has made it easier to work are:

* **Branching Capabilities:**  
    
  - Due to its sophisticated branching capabilities, developers can easily work on multiple branches for the different features of the project.  
  - It also has an easier merge option along with an efficient work-flow feature diagram for tracking it.
* **Distributed manner of development:**  
    
  - Git is a distributed system and due to this nature, it became easier to trace and locate data if it's lost from the main server.  
  - In this system, the developer gets a repository file that is present on the server. Along with this file, a copy of this is also stored in the developer’s system which is called a local repository.  
  - Due to this, the scalability of the project gets drastically improved.
* **Pull requests feature:**  
    
  - This feature helps in easier interaction amongst the developers of a team to coordinate merge-operations.  
  - It keeps a proper track of the changes done by developers to the code.
* **Effective release cycle:**  
    
  - Due to the presence of a wide variety of features, git helps to increase the speed of the release cycle and helps to improve the project workflow in an efficient manner.

**11. How will you create a git repository?**

* Have git installed in your system.
* Then in order to create a git repository, create a folder for the project and then run git init.
* Doing this will create a .git file in the project folder which indicates that the repository has been created.

**12. Tell me something about git stash?**

Git stash can be used in cases where we need to switch in between branches and at the same time not wanting to lose edits in the current branch. Running the git stash command basically pushes the current working directory state and index to the stack for future use and thereby providing a clean working directory for other tasks.

**13. What is the command used to delete a branch?**

* To delete a branch we can simply use the command git branch –d [head].
* To delete a branch locally, we can simply run the command: git branch -d <local\_branch\_name>
* To delete a branch remotely, run the command: git push origin --delete <remote\_branch\_name>
* Deleting a branching scenario occurs for multiple reasons. One such reason is to get rid of the feature branches once it has been merged into the development branch.

**14. What differentiates between the commands git remote and git clone?**

git remote command creates an entry in  git config that specifies a name for a particular URL. Whereas git clone creates a new git repository by copying an existing one located at the URL.

**15. What does git stash apply command do?**

* git stash apply command is used for bringing the works back to the working directory from the stack where the changes were stashed using git stash command.
* This helps the developers to resume their work where they had last left their work before switching to other   
  branches.

**16. Differentiate between git pull and git fetch.**

| **git pull** | **git fetch** |
| --- | --- |
| This command pulls new changes from the currently working branch located in the remote central repository. | This command is also used for a similar purpose but it follows a two step process:  1. Pulls all commits and changes from desired branch and stores them in a new branch of the local repository.  current 2. For changes to be reflected in the current / target branch, git fetch should be followed by git merge command. |

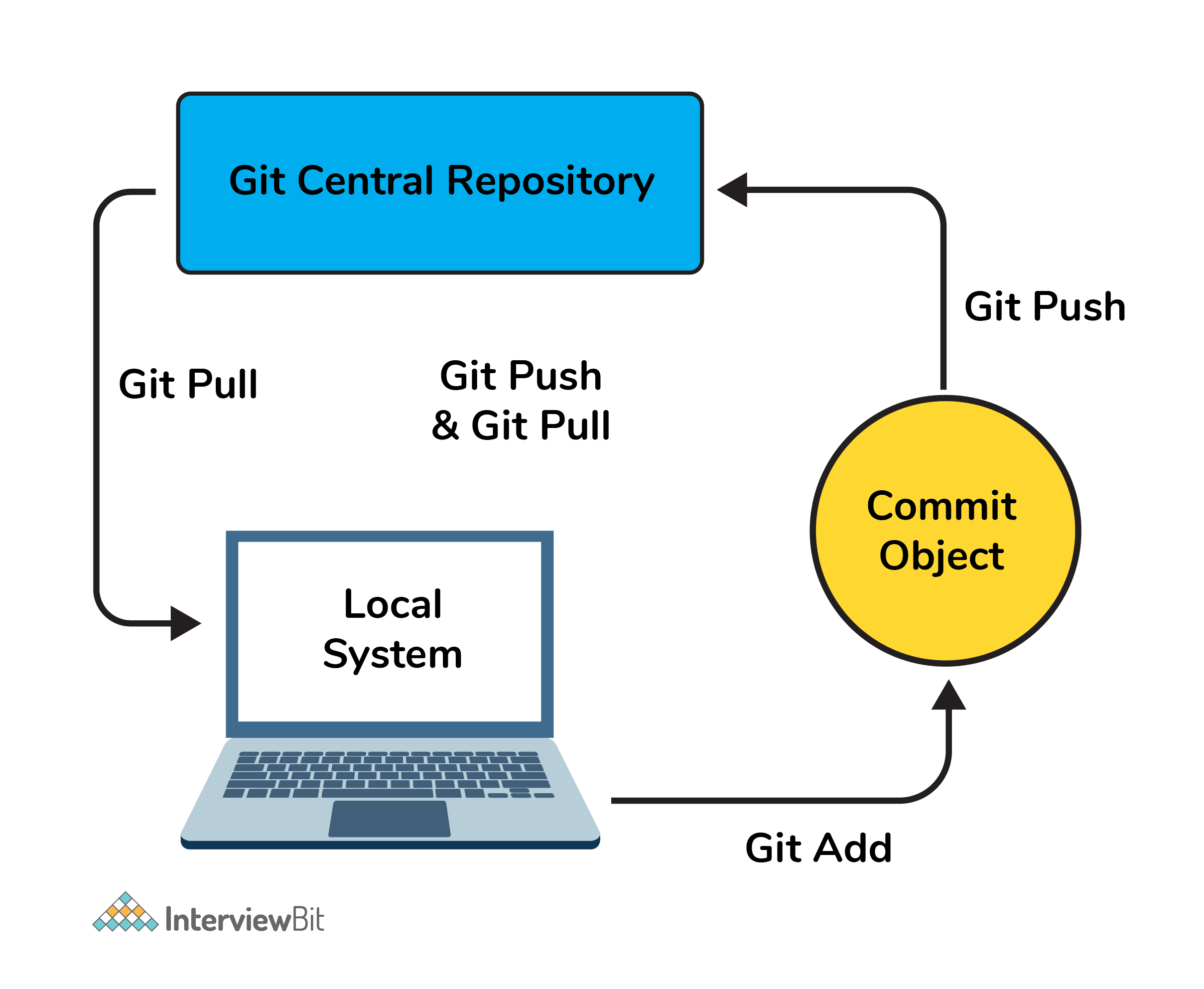
git pull = git fetch + git merge

**17. Can you give differences between “pull request” and “branch”?**

| **pull request** | **branch** |
| --- | --- |
| This process is done when there is a need to put a developer’s change into another person’s code branch. | A branch is nothing but a separate version of the code. |

**18. Why do we not call git “pull request” as “push request”?**

* “Push request” is termed so because it is done when the target repository requests us to push our changes to it.
* “Pull request” is named as such due to the fact that the repo requests the target repository to grab (or pull) the changes from it.



**19. Can you tell the difference between Git and GitHub?**

| **Git** | **GitHub** |
| --- | --- |
| This is a distributed version control system **installed on local machines** which allow developers to keep track of commit histories and supports collaborative work. | This is a **cloud-based source code repository** developed by using git. |
| This is maintained by “The Linux Foundation”. | This was acquired by “Microsoft” |
| SVN, Mercurial, etc are the competitors | GitLab, Atlassian BitBucket, etc are the competitors. |

GitHub provides a variety of services like forking, user management, etc along with providing a central repository for collaborative work.  
  
  
**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. |

**Basic Git Interview Questions and Answers**

**1. What is Git?**

Git is a decentralized version control system engineered for rapid and effective management of projects ranging from modest to vast in scale. It supports collaborative efforts among developers by monitoring file modifications and enhancing teamwork.

**2. What is a repository in Git?**

A Git repository is a location where your project resides, acting as a storage area. This repository can exist locally within a directory on your computer or on a cloud-based platform like GitHub. It encompasses all the files associated with the project, along with a record of the modifications made to these files over time.

**3. What is the difference between Git and GitHub?**

Git is a tool for version control that enables you to monitor and record the evolution of your source code. GitHub is an online hosting service that facilitates the management of Git repositories. GitHub offers a user-friendly web interface along with functionalities such as permission management, task organization, error tracking, and the capability to handle feature suggestions.

**4. How does Git work?**

Git works by taking snapshots of a project's files. Unlike other version control systems, Git records the entire contents of each file and its changes every time a commit is made. This makes operations like branching, merging, and reverting changes more efficient.

**5. What is a commit in Git?**

In Git, a commit is a process that records a version of the project's presently prepared modifications. This record includes details on the modifications implemented, a distinctive identifier (a SHA-1 hash), the creator's identity, and the timestamp of the commit.

**6. What is branching in Git?**

In Git, branching allows you to veer off from the primary development path and proceed with separate tasks without impacting the main workflow. This technique enables the isolated development of features, bug fixes, or experimentation within a specific section of the repository, ensuring that each process remains distinct from the others.

**7. What is a merge in Git?**

Merging is a Git operation that integrates changes from one branch into another. It can be a fast-forward merge, where the target branch is updated to the latest commit of the source branch, or a three-way merge, where divergent branch histories are combined into a new commit.

**8. What is a conflict in Git?**

A conflict in Git occurs when two branches have made edits to the same line in a file or when one branch deletes a file while the other branch modifies it. Git cannot automatically resolve these changes; the developer must manually resolve the conflicts.

**9. What is a pull request?**

A pull request serves as a mechanism for contributing to a project, typically utilized in projects hosted on GitHub. In this process, a developer implements modifications within their own branch, uploads these changes to a repository, and then initiates a pull request. This action prompts the project's maintainers to examine the proposed changes, engage in discussions about possible adjustments, and ultimately integrate the pull request into the primary branch.

**10. What is `git fetch` vs. `git pull`?**

`git fetch` downloads updates from a remote repository to your local repository without integrating them. On the other hand, `git pull` not only fetches the updates but also incorporates them into your active branch.

**11. How do you revert a commit that has already been pushed and made public?**

To undo the modifications introduced by a previous commit while ensuring safety for public commits, employ `git revert <commit\_hash>`, which generates a fresh commit that reverses the earlier changes. On the other hand, `git reset` allows reverting to an earlier state; however, exercise caution when applying it to public commits, as it alters the commit history.

**12. What is a `.gitignore` file?**

A `.gitignore` file is a textual document instructing Git on the files or directories to exclude from a project. This commonly encompasses files created during the build phase, local setup files, or files with confidential data.

**13. How do you clone a repository?**

To create a local copy of a repository on your machine, execute the command `git clone <repository\_url>`. This action duplicates the repository locally.

**14. What is `git stash`?**

The `git stash` command temporarily stores your working directory's modifications, allowing you to switch branches without discarding your current progress.

**15. How do you view the commit history?**

Use `git log` to view the commit history. There are many options to customize the output, such as `git log --oneline` for a condensed view.

**16. What is a remote in Git**?

A Git remote refers to a shared repository utilized by all team members for the purpose of exchanging their updates. Typically, this repository is on a server or a cloud-based hosting service like GitHub.

**17. How do you create a new branch?**

Use `git branch <branch\_name>` to create a new branch. Use `git checkout -b <branch\_name>` to create and switch to it in one step.

**18. What is `git merge --squash`?**

The `git merge --squash` command consolidates all commits from a specified feature branch into a single commit when merging into the target branch, resulting in a tidier project history.

**19. How do you resolve a merge conflict?**

To resolve a merge conflict, edit the files to fix the conflicting changes. Then, use `git add` to stage the resolved files and `git commit` to commit the resolved merge.

**20. What is `git rebase`?**

`git rebase` transfers modifications from one branch to another, enabling the creation of a streamlined project timeline by relocating the updates of a feature branch to the forefront of the main branch.

**21. What is the difference between `git merge` and `git rebase`?**

The main difference is in how the branch history is presented. `git merge` preserves the history of a feature branch by creating a new merge commit. `git rebase` rewrites the feature branch's history to appear as if it was developed from the latest main branch, creating a linear history.

**22. How do you change the last commit?**

Use `git commit --amend` to modify the most recent commit. This can change the commit's message or include new changes.

**23. What is `git push`?**

`git push` is used to upload local repository content to a remote repository. It transfers commits from your local repo to the remote.

**24. How do you delete a branch?**

Use `git branch -d <branch\_name>` to delete a local branch. If the branch is not fully merged, you may need to use `-D` instead. To delete a remote branch, use `git push <remote\_name> --delete <branch\_name>`.

**25. What is `git checkout`?**

`git checkout` allows navigating between different branches or reverting working tree files to a previous state. However, in the latest versions of Git, it is advised to use `git switch` for changing branches and `git restore` to revert files, each designated for these specific functions.

**26. How do you list all the remote repositories configured?**

Use `git remote -v` to list all the remote repositories configured for your local repository.

**27. How do you add a file to the staging area?**

Use `git add <file\_name>` to add a file to the staging area, making it ready for a commit.

**28. How do you remove a file from Git but not delete it from your file system?**

Use `git rm --cached <file\_name>` to remove a file from Git without deleting it from your filesystem.

**29. What is `git diff`?**

`git diff` shows the differences between files in the working directory and the index, or between commits.

**30. How do you rename a Git branch?**

To rename the current branch, use `git branch -m <new\_name>`. To rename a different branch, use `git branch -m <old\_name> <new\_name>`.

**31. What does `git reset` do?**

`git reset` is used to undo changes. It has three main modes: `--soft`, `--mixed`, and `--hard`.

**32. How do you amend a commit message?**

Use `git commit --amend` to change your most recent commit message.

**33. What is the HEAD in Git?**

HEAD is a reference to the last commit in the currently checked-out branch.

**34. How do you find a commit by a message?**

Use `git log --grep=<search-pattern>` to search through commit messages.

**35. What is the difference between `git stash pop` and `git stash apply`?**

`git stash pop` applies stashed changes and removes them from the stash. `git stash apply` applies stashed changes but keeps them in the stash.

**36. How do you list all branches that contain a specific commit?**

Use `git branch --contains <commit>`.

**37. What is a fast-forward merge in Git?**

A fast-forward merge happens when the target branch's tip is behind the merged branch's tip, allowing the target branch to "catch up" by just moving forward to the merged branch's tip.

**38. How do you create an empty commit?**

Use `git commit --allow-empty` to create a commit with no changes.

**39. How do you switch branches in Git?**

Use `git checkout <branch-name>` to switch branches.

**40. How do you ignore changes in a tracked file?**

Use `git update-index --assume-unchanged <file>` to ignore changes in a tracked file temporarily.

**Intermediate Git Interview Questions and Answers**

**1. Explain the Git branching strategy you use.**

A common strategy is the Git Flow, which involves having a master branch, develop branch, feature branches, release branches, and hotfix branches, each serving a different purpose in the development cycle.

**2. What is the significance of `git merge --no-ff`?**

`git merge --no-ff` creates a merge commit even if the merge could be resolved as a fast-forward, preserving the history of a feature branch.

**3. How do you revert a Git repository to a previous commit?**

Use `git reset --hard <commit-hash>` to revert to a specific commit, discarding all changes since that commit.

**4. What is a detached HEAD in Git?**

A detached HEAD occurs when you check out a commit, branch, or tag that is not the latest commit of a branch.

**5. How do you fix a detached HEAD?**

Create a new branch from the detached HEAD state with `git branch <new-branch>` and check it out with `git checkout <new-branch>` to move back to a non-detached state.

**6. Explain the difference between `git pull` and `git fetch` followed by `git merge`.**

`git pull` does a `git fetch` followed by a `git merge` automatically. Using `git fetch` followed by `git merge` allows you to review changes before merging.

**7. What is `git rebase --interactive`?**

`git rebase --interactive` allows you to modify commits in many ways, such as rewriting, combining, and removing commits in a more controlled manner.

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

Use `git rebase --interactive HEAD~N` and choose `squash` for the commits you want to combine.

**9. What are submodules in Git?**

Submodules enable the inclusion of one Git repository within another as a subdirectory. This feature is beneficial for integrating external projects or libraries into your main project.

**10. How do you update a submodule?**

Use `git submodule update --remote` to fetch and update your submodules.

**11. What is `git bisect`? How do you use it?**

`git bisect` assists in identifying the commit responsible for introducing a bug through the application of a binary search algorithm.

**12. How do you change the URL of a remote repository?**

Use `git remote set-url <remote-name> <new-url>` to change the URL.

**13. What is the significance of `git push --force`?**

`git push --force` is used to overwrite the remote history with your local history. It should be used with caution as it can overwrite changes in the remote repository.

**14. How do you clean untracked files from your working directory?**

Use `git clean` to remove untracked files from your working directory.

**15. What is `git reflog`?**

`git reflog` shows a log of where the HEAD and branch references have been, allowing you to navigate back to previous states.

**16. How do you resolve a rebase conflict?**

Resolve the conflict manually in the affected files, then use `git add` to stage the resolved files, and continue the rebase with `git rebase --continue`.

**17. What is a bare repository in Git?**

A bare repository is a Git repository that does not have a working directory, making it suitable for sharing code as it contains only the version history.

**18. How do you rename a remote branch?**

Rename the local branch, push it to the remote, and then delete the old remote branch.

**19. What is the purpose of `git tag -a`?**

`git tag -a` creates an annotated tag, which includes metadata such as the tagger name, email, and date, useful for marking releases.

**20. How do you find a list of files that have changed in a specific commit?**

Use `git show --name-only <commit-hash>` to list the files that changed in a commit.

**21. What is `git blame` and how do you use it?**

`git blame` shows what revision and author last modified each line of a file. It's useful for tracking changes and identifying who made them.

**22. How do you configure Git to ignore changes in file permissions?**

Use `git config core.fileMode false` to ignore file permission changes.

**23. What is the difference between `HEAD`, `working tree` and `index` in Git?**

`HEAD` refers to the last commit on the current branch, `working tree` is the set of files in your directory, and `index` (or staging area) is a staging area for commits.

**24. How do you make an existing Git branch track a remote branch?**

Use `git branch --set-upstream-to=<remote>/<branch> <local-branch>` to set a local branch to track a remote branch.

**25. What does `git fetch --prune` do?**

`git fetch --prune` removes remote-tracking branches that no longer exist on the remote.

**26. How do you combine multiple commits into one without merging?**

Use `git rebase --interactive` to squash commits into a single commit without creating a merge commit.

**27. What is `git stash drop`?**

`git stash drop` removes a single stashed state from the stash list.

**28. How do you list all the remote branches?**

Use `git branch -r` to list all remote branches.

**29. What is the purpose of `git gc` (garbage collection)?**

`git gc` cleans up unnecessary files and optimizes the local repository.

**30. How do you find who introduced a line of code using Git?**

Use `git blame <file-name>` to see who last modified each line of a file.

**31. What does `git commit --dry-run` do?**

It simulates a commit, showing what would be committed without actually committing the changes.

**32. How do you revert changes made to the working directory?**

Use `git checkout -- <file-name>` to discard changes in the working directory.

**33. What is the purpose of `git log --graph`?**

`git log --graph` displays the commit history in a graphical representation.

**34. How do you list all tags in Git?**

Use `git tag` to list all tags in the current repository.

**35. What is `git show` and how do you use it?**

`git show <commit-hash>` displays the information about a git object like a commit.

**36. How do you copy a commit from one branch to another?**

Use `git cherry-pick <commit-hash>` to apply the changes from a commit on another branch to the current branch.

**37. What is `git archive`?**

`git archive` is used to create an archive (zip or tar) of files from a named tree.

**38. What does `git checkout --track <remote/branch>` do?**

It creates a new branch that tracks the specified remote branch.

**39. How do you compare two branches in Git?**

Use `git diff <branch1>..<branch2>` to see the differences between the two branches.

**40. What is `git reset --soft`?**

`git reset --soft <commit-hash>` undoes commits but keeps the changes in the staging area.

**Popular Git Interview Questions and Answers**

**1. What is Git?**

Git is a decentralized system for version control that enables developers to monitor and control modifications to their code repository.

**2. How do you clone a repository?**

Use `git clone <repository-url>` to make a copy of the target repository on your local machine.

**3. `git pull` vs `git fetch`**

`git pull` updates your current branch with the latest changes from the remote, while `git fetch` retrieves the latest changes from the remote without integrating them into your local branch.

**4. Explain the Git workflow.**

The basic Git workflow involves creating branches, making changes, committing those changes, and then merging those changes back into the main branch.

**5. How do you fix a merge conflict?**

Fix merge conflicts by editing the conflicted files to choose which changes to keep, then staging and committing those changes.

**6. What is a branch in Git?**

A branch in Git is a separate line of development, allowing you to work on different features or fixes independently.

**7. How do you create a new branch and switch to it?**

Use `git checkout -b <branch-name>` to create and switch to a new branch.

**8. What is a commit in Git?**

A commit represents a specific moment's capture of your repository, documenting the modifications made to your project.

**9. How do you push changes to a remote repository?**

Use `git push <remote-name> <branch-name>` to send your committed changes to a remote repository.

**10. What is `git merge` and how do you use it?**

`git merge <branch>` merges changes from one branch into the current branch.

**11. What is a `.gitignore` file?**

A `.gitignore` file identifies files that should remain untracked and be disregarded by Git on purpose.

**12. How do you revert a commit?**

To create a new commit that reverses the changes introduced in a specific commit, execute the command `git revert <commit-hash>`.

**13. What is a fast-forward merge?**

A fast-forward merge happens when the target branch's head is behind the merged branch's head, allowing the target branch to fast-forward to the tip of the merged branch.

**14. How do you change a commit message that you have already pushed?**

Use `git commit --amend` to change the last commit message, then use `git push --force` to update the remote repository.

**15. What is the difference between `git checkout`, `git reset`, and `git revert`?**

`git checkout` switches branches or restores working tree files, `git reset` changes the head to a specific state, and `git revert` undoes changes by creating a new commit.

**16. How do you squash commits?**

Use `git rebase -i` and then choose to squash the commits in the interactive prompt.

**17. What is `git stash`?**

`git stash` temporarily shelves changes so you can work on a different branch with a clean working directory.

**18. How do you list all the remote connections for a repository?**

Use `git remote -v` to list all remote connections.

**19. What does `git fetch` do?**

`git fetch` updates your local copy of a remote branch, without merging the changes into your current branch.

**20. How do you delete a branch locally and remotely?**

Locally: `git branch -d <branch-name>`, remotely: `git push <remote-name> --delete <branch-name>`.

**21. What is the purpose of `git config`?**

`git config` is used to set configuration options for your Git installation, such as user name and email.

**22. How do you list all the branches that are merged into the current branch?**

Use `git branch --merged` to list branches merged into the current branch.

**23. What is `git log` and how do you use it?**

`git log` displays the commit history. You can use various options to format the output, such as `--oneline`, `--graph`, etc.

**24. How do you find a specific commit by message?**

Use `git log --grep="commit message"` to search the commit history for a specific message.

**25. What is the use of `git diff`?**

`git diff` shows the differences between commits, commit and working directory, etc.

**26. How do you add files to a commit?**

Use `git add <file-name>` to stage a file for commit.

**27. What does `git commit -m "message"` do?**

It commits the staged changes to the repository with a message describing the commit.

**28. How do you update a Git repository to the latest version?**

Execute the command `git pull` to retrieve and integrate updates from the remote repository into your local branch.

**29. What is `git branch -d` and how do you use it?**

`git branch -d <branch-name>` deletes a local branch, if it has been fully merged in its upstream branch.

**30. How do you see the changes made by a specific commit?**

Use `git show <commit-hash>` to display the changes made in a commit.

**31. What is the significance of the HEAD pointer in Git?**

HEAD points to the most recent commit on the current branch, indicating the workspace's latest status.

**32. How do you rename a local Git branch?**

Use `git branch -m <old-name> <new-name>` to rename a local branch.

**33. What is the purpose of `git checkout -- <file>`?**

It is used to discard changes in the working directory for the specified file.

**34. How do you create a tag in Git?**

Use `git tag <tag-name>` to create a lightweight tag, or `git tag -a <tag-name> -m "message"` for an annotated tag.

**35. What is `git push --tags`?**

It pushes all your tags to the remote repository.

**36. How do you revert to a previous commit without losing the changes made since?**

Use `git revert <commit-hash>` for each commit you want to revert. This creates new commits that reverse the changes.

**37. What is the use of `git rm`?**

`git rm <file>` removes files from the working directory and stages the deletion.

**38. How do you compare two commits?**

Use `git diff <commit1> <commit2>` to see the differences between two commits.

**39. What is `git rebase` and how is it different from merge?**

`git rebase` rewrites the commit history by moving the branch to the tip of the target branch, whereas merge combines two histories together.

**40. How do you handle a merge conflict in Git?**

Resolve the conflict manually in the affected files, mark them as resolved with `git add`, and then complete the merge with `git commit`.

**Advanced Git Interview Questions and Answers**

**1. What is the Git object model?**

The Git object model consists of blobs (representing file data), trees (which organize the file structure), commits (which record the changes), and tags (which mark specific points in history).

**2. Explain the difference between `git merge` and `git rebase` and when you would use each.**

`git merge` integrates changes from one branch to another, preserving the history of both branches. `git rebase` rewrites the history by placing commits from one branch onto another, creating a linear history. Use `merge` to preserve the history of a feature branch, and `rebase` to clean up the commit history before merging.

**3. How does Git store data?**

Git preserves project data by taking snapshots at various points in time. Every commit acts as a record of the files' condition at that specific instance. To manage this information effectively, Git employs both delta compression and the use of direct object references.

**4. What is the role of the `.git/index` file?**

The `.git/index` file acts as the staging area (or index) for Git. It tracks which changes are staged for the next commit.

**5. Explain what a "detached HEAD" is and how you might end up in one.**

A detached HEAD occurs when you check out a specific commit rather than a branch. This temporary state allows you to navigate through the history of the repository without making changes to the branches.

**6. How do you interactively rebase the last N commits?**

Use `git rebase -i HEAD~N` where N is the number of commits you want to rebase. This opens an editor showing the last N commits and allows you to reorder, squash, edit, or drop commits.

**7. What are the advantages of using a rebase over a merge?**

Rebase creates a cleaner, linear commit history, which can simplify the understanding and exploration of project history. It avoids unnecessary merge commits and can make the history easier to navigate.

**8. What is a git hook and how might you use it?**

Git hooks are automated scripts triggered before or after specific Git commands, such as pre-commit, pre-push, post-commit, and post-receive, are executed. They serve various functions, including syntax verification, testing, and the enforcement of project guidelines.

**9. How do you squash the last N commits into one using Git?**

Use `git rebase -i HEAD~N` and mark all but the first commit with "squash" or "fixup" to combine them into a single commit.

**10. What is the Git Garbage Collection and when is it called?**

Git's garbage collection (`git gc`) is a housekeeping task that cleans up unnecessary files and optimizes the local repository. It is automatically called by certain commands but can be manually triggered for maintenance.

**11. How do you find and restore a deleted file in the Git history?**

Use `git log -- <file-path>` to find the commit where the file was deleted, then use `git checkout <commit-hash>^ -- <file-path>` to restore it.

**12. What is the significance of `git push --force-with-lease` over `git push --force`?**

`git push --force-with-lease` ensures that you do not overwrite any work on the remote repository that you haven't seen, unlike `git push --force` which overwrites the remote changes blindly.

**13. How can you achieve a Git bisect manually?**

Manually performing a bisect involves checking out commits in a binary search manner to narrow down the commit that introduced a bug, but this is automated with `git bisect`.

**14. Explain the difference between `git stash pop` and `git stash apply`.**

`git stash pop` implements the modifications from the most recent stash and then deletes it from the stash inventory, while `git stash apply` enacts the changes yet preserves them within the stash inventory.

**15. How do you reapply a commit that has been reverted?**

Use `git revert <commit-hash>` to revert the revert commit, effectively reapplying the original commit.

**16. What is the purpose of `git cherry-pick`?**

`git cherry-pick` is used to apply the changes introduced by some commits from one branch onto another branch.

**17. How do you resolve a binary file conflict in Git?**

Binary file conflicts must be resolved manually by choosing which version of the file to keep, then adding and committing that file.

**18. What is the function of `git blame -L`?**

`git blame -L` shows who last modified each line of a file within a given line range.

**19. How do you handle large files with Git?**

Use Git Large File Storage (LFS) to handle large files by storing references in the repository but keeping the actual files on a separate server.

**20. What is the use of `git submodule` and how do you update one?**

Git submodules enable you to maintain a Git repository as a subdirectory within a different Git repository. To update a submodule, use the command `git submodule update --remote`.

**21. How do you list all the aliases you have set in Git?**

Use `git config --get-regexp alias` to list all aliases set in the Git configuration.

**22. Explain how to change the base branch of a pull request.**

Modifying the base branch of a pull request usually requires altering the target branch through the pull request interface provided by a Git hosting platform (such as GitHub or GitLab).

**23. What does `git reflog` show and how can it be useful?**

`git reflog` shows a log of the changes to the local repository's HEAD. It can be useful for recovering lost commits or exploring changes made to branches.

**24. How do you integrate changes from a remote branch without a merge commit?**

Use `git pull --rebase` to rebase the current branch on top of the remote branch, integrating changes without a merge commit.

**25. What is a symbolic reference in Git?**

A symbolic reference is a reference to another reference, such as a branch name being a reference to a commit. The HEAD is a common example of a symbolic reference.

**26. How do you find which commit a bug was introduced in?**

Use `git bisect` to perform a binary search through the commit history to find the commit that introduced a bug.

**27. What is the purpose of `git worktree`?**

`git worktree` allows you to have multiple working trees attached to the same repository, enabling you to work on multiple branches simultaneously without switching the current worktree.

**28. How do you configure Git to use a proxy?**

Configure Git to use a proxy by setting the `http.proxy` or `https.proxy` configuration, for example, `git config --global http.proxy proxy-url`.

**29. What is the difference between a shallow clone and a deep clone in Git?**

A shallow clone (`git clone --depth 1`) creates a copy of the repository with a limited history depth, reducing time and space, whereas a deep clone includes the complete history.

**30. How do you amend the author of a previous commit?**

Use `git commit --amend --author="New Author Name <email@domain.com>"` to change the author of the most recent commit, or use an interactive rebase for older commits.

**31. What is the purpose of `git fsck` and how do you use it?**

`git fsck` (file system check) is used to verify the integrity of the Git file system, checking for corrupt objects.

**32. How do you create a patch with Git?**

Use `git diff > patch\_name.patch` to create a patch file with the changes between commits or branches.

**33. What is the use of `git revert --no-commit`?**

`git revert --no-commit` reverts the changes made by one or more commits without committing the revert, allowing you to make additional changes before committing.

**34. How do you implement a Git workflow in a team environment?**

Implement a Git workflow by establishing a branching model (like Git Flow or GitHub Flow), setting up a code review process, defining commit message guidelines, and integrating CI/CD pipelines.

**35. What are Git hooks and how can they be customized for a project?**

Git hooks are executable scripts triggered by Git prior to or following events like commits, pushes, and receptions. These scripts can be tailored by placing them in the `.git/hooks` directory.

**36. How do you troubleshoot connectivity issues when using Git with a remote repository?**

Troubleshoot connectivity issues by checking network connections, verifying remote repository URLs, ensuring authentication credentials are correct, and reviewing proxy configurations.

**37. What is the significance of the `git fetch --tags` command?**

`git fetch --tags` fetches all tags from the remote repository, updating your local tag references.

**38. How can you use Git to track changes in any file, even if it's not code?**

Git can track changes in any file type (binary or text) by adding the file to the repository and committing changes. For binary files, it's often beneficial to use Git LFS.

**39. Explain the process of contributing to an open-source project using Git.**

Contributing to an open-source project typically involves forking the project repository, cloning your fork, making changes in a new branch, pushing the branch to your fork, and submitting a pull request to the original repository.

**40. How do you manage multiple configurations for different projects in Git?**

Manage multiple configurations by using the `git config` command with the `--global`, `--system`, or `--local` flags to set configuration options at different levels or by using includeIf in the Git configuration to include specific configurations based on the repository path.

**Q1. What is Git and how is it used in DevOps?**

Git is a DevOps tool for source code management. It is a free and open-source version control system used to handle small to very large projects efficiently. Git is used to track changes in the source code, enabling multiple developers to work together on non-linear development.

**Q2. Mention the most commonly used Git commands and their functions**

Your interviewer can ask you about the following commands individually, or you may have to make comparisons between them. You must prepare this crucial Git interview question thoroughly. The following table enumerates some key commands and their functions:

**Git Commands and Functions**

* **git config:**It configures the username and email address.
* **git add:**This command adds one or more files to the staging area.
* **git add <file\_name>:**It specifically adds files one by one.
* **git init:**This command initializes an empty Git repository.
* **git commit:**It allows you to commit changes to the head but not to the remote repository.
* **git stash:**It provides a clean working directory for other tasks while pushing the current working directory state and index to the stack for future use.
* **git stash drop:** It deletes the stashed element (last-added stash item by default)  from the directory. Can also delete a specific topic included as an argument.
* **git stash pop:** It discards the specified stash (topmost stash by default) after applying it.
* **git pull:**It pulls innovation or commits from a specific branch of your central repository. It also updates your object branch in your local repository.
* **git fetch:** It pulls all new commits from the desired branch. It then saves it in a new branch in your local repository.
* **git merge:**It reflects the Git fetch changes in your target branch. You can use Git merge and specify the name of the other branch to bring it into the target branch.
* **git branch – merged:** This command records the branches merged into the present branch.
* **git reset –mixed:**You can use it for undoing changes of the working directory and the Git index.
* **git merge –abort:**It stops the merge process and returns back to the state before the merging occurs.
* **git revert:**It creates a new commit to undo the changes of the previous commit. This command adds a new history without modifying the existing one.
* **git reset:**You can use this command for undoing the local changes done in the Git repository. It operates on the Git index, commit history, and the working directory.
* **git status:** It shows the difference between the working directory and the index.
* **git clone:**It generates a copy of a current Git repository.
* **git remote:**It helps you create, view, and delete remotes associated with the local repository.
* **git branch –d [head]:** It simply deletes a branch.
* **git branch -d <local\_branch\_name>:** You can use this command to delete a branch locally.
* **git push origin –delete <remote\_branch\_name>:** You can use this command to delete a branch remotely.
* **git reflog:** It tracks every single change made in the repository references and maintains the branches/tags log history, and thus gets the name reflog (reference + log).
* **git annotate [<options>] <file> [<revision>]:** It annotates each line with information from the commit that introduced that change. It can also optionally annotate from a given revision.
* **git cherry pick:**It introduces certain commits from one branch into another branch within the repository.
* **git bisect:**It performs a binary search and detects the commit that introduced a bug or regression in the project’s history.

**Beginner Git Interview Questions**

Here are some basic Git interview questions that you can practice for your upcoming interview:

1. How is Git different from SVN?
2. What are the advantages of using Git?
3. What are the limitations of Git?
4. Are Git and GitHub the same?
5. What is a Git repository?
6. How can you create a Git repository?
7. What is a bare repository?
8. Why do we require branching in Git?
9. What do you understand about the “Index” and “Staging Area” in GIT?
10. What is a conflict in Git? How will you resolve it?

**Advanced Git Interview Question and Answers**

**Q1. Why is it better to create an additional commit than amending an existing one?**

You must consider the following reasons while answering this Git interview question:

* Amending an existing commit can potentially overwrite or lose important information from the recent commit. If you only change the commit message, it won’t create a problem. However, if you modify the contents, there are more chances of excluding something important.
* If you abuse “git commit-amends,” it can cause a small commit to increase and acquire inappropriate changes.

**Q2. Explain the different types of branching systems in Git.**

This Git interview question tests your branching knowledge with Git. You must elaborate on how you have utilized branching in your past activity. You can consider the following points while answering such Git interview questions:

* **Feature branching:**A component branch model stores the majority of the changes for a specific element within a branch. The branch is merged into the master when the item is tested and approved by automation.
* **Task branching:**This model allows each assignment to be actualized on its branch with the undertaking key included in the branch name.
* **Release branching:**You can clone the create branch that has procured enough features for a discharge to frame a release branch and begin the following discharge cycle. You cannot add new features after this point. Only bug fixes, documentation generation, and other release-oriented tasks are allowed in this branch.

**Q2. What is Subgit? Why would you use it?**

The tool that migrates SVN to Git is termed Subgit. It effectively detects the settings of your remote SVN repository and downloads SVN revisions. It then converts them to Git commits, thereby emerging as a stable solution for a company-wide migration from SVN to Git. The primary reasons for using Subgit are as follows:

* It is superior to git-svn.
* You need not change the infrastructure.
* It allows you to use all Git and all sub-version features.
* It provides a stress-free migration experience.

**Q3. What are the advantages of forking workflow?**

This is one of the most frequently asked Git interview questions. There are several advantages of the forking workflow over other popular Git workflows, and you can compare them. You can highlight the following points while answering such Git interview questions.

* It gives you your own server-side repository rather than using a single server-side as the “central” codebase.
* It is ideal for public open-source projects.
* You can integrate contributions without requiring everybody to push to a single central repository that gives clean project history. You can push to your own server-side repositories as a developer, while the project maintainer can push to the official repository.

**Advanced Git Interview Questions**

Given below are some of the advanced Git interview questions for your practice:

1. What are the various Git repository hosting functions?
2. What is git is-tree?
3. What are “hooks,” and what do they comprise of in Git?
4. What is GitLab?
5. What is DAG in Git?
6. What is the role of Git pull origin master?
7. What is the difference between fork and branch?

**Scenario-based Git Interview Questions**

1. How would you return a commit that has just been pushed and made open?
2. How will you know if a branch has just been combined into a master in Git?
3. How would you find a commit that broke something after a merge operation?
4. What would you do to squash the last N commits into a single commit?
5. How would you remove a file from Git without removing it from your file system?
6. When would you choose “git rebase” instead of “git merge”?
7. How can you configure a Git repository to run code sanity checking tools before making commits? How will it prevent them if the test fails?
8. One of your team members accidentally deleted a branch and pushed the changes to the central Git repository. How would you recover this branch, given that there are no other Git repositories and no other teammates have a local copy?
9. How would you copy a commit made in one branch to another branch?
10. How would you set up a script to run every time a repository gets new commits through push?

**Basic Git Interview Questions for Freshers**

**1. What is the process for creating a repository in Git?**

If we want to create a repository in Git, then we need to run the command “git init”. With this command .git repository, we can create a directory in the project directory.

**2. What is origin in Git?**

Origin refers to the remote repository that a project was originally cloned from and is used instead of the original repository’s URL. This allows for easier referencing.

**3. What is the git push command?**

The git push command is applied for uploading content to a remote repository from a local repository. Pushing can overwrite changes, so it should be used with caution.

**4. What is the git pull command?**

The git pull command is for fetching and downloading content from a remote repository and integrating it with a local repository.

**5. What is the difference between git rebase and git merge?**

In git rebase, a feature branch is moved into a master. Git merge maintains the history by adding a new commit.

**6. Why do we need branching in GIT?**

With the help of branching, we can have our own branch and we can also jump between various branches. We can go to our previous work, at the same time keeping our recent work intact.

**7. What is the difference between Git and GitHub?**

Git is a version control system that is used in the management of the source code history. GitHub, on the other hand, is a cloud-based hosting service that is used in the management of Git repositories. GitHub is designed to help in the better management of open-source projects.

**8. State a way to create a new branch in Git.**

If we want to create a new feature into the main branch, then we can use the command ‘git merge’ or ‘git pull’.

**9. What is the purpose of the .gitignore file?**

The purpose of .gitignore files is to ensure that certain files that are not tracked by Git should remain untracked.

**10. How do you clone a specific branch from a Git repository?**

To clone a specific branch from a repository, you can use git clone -b <branch> <repository-url>.

**11. How do you rename a local Git branch?**

To rename a local git branch, you can use git branch -m <old name> <new-name>. If already on the branch that has to be renamed, write git branch -m <new-name>

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

To find the list of files that have changed in a particular commit, you can use git show –name-only <commit-hash>.

**13. How do you remove a file from the staging area in Git?**

To remove a file from the staging area, you can execute git reset HEAD <file>

**14. What are the advantages of using GIT?**

* Data redundancy and data replication is possible
* It is a highly available service
* For one repository we can have only one directory of Git
* The network performance and disk utilization are excellent
* It is very easy to collaborate on any project
* We can work on any sort of project within the Git

**15. What is the difference between git init and git clone?**

git init command is used to create a new repository locally, whereas the git clone command copies an existing repository from some other location.

**16. What is the purpose of a .gitkeep file?**

gitkeep is not a git feature but is a convention to keep an empty folder in the repository.

**17. What is the use of the git diff command?**

The git diff command helps the users see, compare, and understand the changes in the project.

**18. What is Git Bash?**

Git Bash is an application that installs Bash, Git, and a few Bash utilities that are commonly used on a Windows OS. In Git Bash, interaction is possible with Git elements and the repository through different commands.

**19. What is the purpose of the git clean command?**

The git clean command removes the untracked files from the working directory.

**20. What is tagging in Git?**

Tagging allows developers to mark all the important checkpoints through the course of their projects’ progress. Instead of commit IDs, tag names can be used while commits are checked out and pushed to a remote repo.

**21. What is forking in Git?**

A repository copy is called a fork. So, forking allows one to experiment with changes without worrying about the original project. This process is ideal for proposing changes to someone else’s projects.

**22. What is the meaning of “Index” or “Staging Area” in GIT?**

When we are making the commits, we can make changes to it, format it and review it in the intermediate area known as ‘Staging Area’ or ‘Index’.

**23. What language is used in GIT?**

C is the programming language that is used for creating Git which ensures that the overheads are reduced.

**24. How do you create a copy of a local branch?**

To create a copy of a local branch, you can use the command git checkout -b <new-branch-name> <existing-branch-name>.

**Intermediate Git Interview Questions**

**25. How to resolve a conflict in Git?**

If we want to resolve a conflict in Git, then we need to edit the files for fixing the conflicting changes and then we can run “git add” to add the resolved files and after that we can run the ‘git commit’ for committing the repaired merge.

**26. What is a Git repository? Name some popular Git hosting services.**

Repositories contain a batch of files that are different versions of a project. These files are imported from the repositories into the local servers of users for further modifications and updates in the content.

**A few popular Git hosting services are:**

GitHub

GitLab

Bitbucket

SourceForge

**27. What are Git and GitHub?**

Git is an open-source and free distributed version control system developed to handle projects of all sizes quickly and efficiently.

GitHub uses Git to provide Internet hosting for version control and software development. It offers the functionality of distributed version control and source code management, which is found in Git, in addition to other unique features.

**28. What is cherry-pick in Git?**

Git cherry-pick is a command that allows the picking of arbitrary Git commits by reference and adding them to the HEAD. Cherry-picking is the process of picking a commit from one branch and applying it to another. It helps in undoing changes.

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

Git fetch retrieves new data from a remote repository but does not integrate it into our working files. It helps in checking if any changes happened in the remote repository. It does not manipulate or destroy anything in the process.

Git pull, on the other hand, updates the HEAD with the latest changes from the remote server and directly integrates it into the working copy files. Using git pull can end in merge conflict as it tries to merge remote changes with the local ones.

**30. Explain git checkout in Git.**

Git checkout allows for the switching of the HEAD. It can be used to restore the historic versions of files as well. The command operates upon files, commits, and branches.

**31. What does git rebase do?**

Rebasing is the reapplying of commits on top of another base trip. A sequence of commits is applied from distinct branches into the final commit. It is a linear process of merging and an alternative to the git merge command. Rebasing makes it seem like one has created a branch from a different commit.

**32. What is revert in Git?**

The git revert command is a forward-moving undo operation. It is a safe way to undo changes as it will create a new commit that inverses the changes instead of deleting or orphaning commits in the commit history.

**33. What is the difference between resetting and reverting?**

While git reset changes the state of the branch to a previous one by removing all of the states after the desired commit, git revert does it through the creation of new reverting commits and keeping the original one intact.

**34. What is the difference between ‘git remote’ and ‘git clone’?**

Git remote adds a reference to a remote repository for further tracking. Git clone, like its name, clones an existing remote repository and builds a new one.

**35. What is GIT stash?**

The Git stash will take the working directory in the current state and index it to put on the stack at a later stage so that what we get is a clean working directory.

**Git Stash**

This means that if we are in the middle of some task and need to get a clean working directory and simultaneously we want to keep all our current edits, then we can use the Git stash.

**36. How are fork, branch, and clone different from each other?**

Forking creates a copy of the original repository, and it remains in the GitHub account. Whereas, in cloning, the repository is copied to the local machine using Git. Forking is used to propose changes to the repository owners. In cloning, the changes are directly pushed to the original repository, provided the user has write access. A branch occurs within a repository and is a way to keep developing and modifying the software without affecting the main project.

**37. What is the difference between git reflog and log?**

The git log is a public record of the commit history for a branch. Reflog, on the other hand, is a private one of the repository’s local commits.

Unlike reflog, the git log is a part of the Git repository and is replicated after a push, fetch, or pull. A developer cannot access a local repository’s reflog without having access to the computer where it is located.

**38. What is GIT stash drop?**

When we are finished with working on the stashed item or want to remove the list, we can use the Git stash drop. This will ensure that the item that is last added by default or any particular item can be removed from the argument.

**39. How to identify if a certain branch has been merged into master?**

Git branch –merged master – shows all branches that are merged into master

Git branch – merged – shows all branches that are merged into the head

Git branch – no-merged –shows all the branches that are not merged

**40. What are the main differences between Git and SVN?**

|  |  |  |
| --- | --- | --- |
| Criteria | Git | SVN |
| Type of version control | Distributed | Centralized |
| Access to networks | Not mandatory | Mandatory |
| Global revision number | Not available | Available |
| Content | Cryptographic SHA-1 Hash | No hashed content |

**Here we list some of the most important differences between Git and SVN:**

When it comes to handling large files, Git is not preferred but SVN can handle multiple projects in the same repository

Git does not have ‘commits’ across multiple branches but SVN lets you create the folders on any location in the repository layout You cannot commit changes in Git but SVN lets you create a tag as a branch and you can create multiple revisions under a root tag

**41. What are the benefits of using a pull request in a project?**

Teams can work on different parts of the system at the same time and then merge their changes together using the pull requests in a project. Therefore, this boosts team efficiency.

**42. What is a 'detached HEAD' state in Git?**

A detached HEAD state occurs when the HEAD does not point to a branch but rather points toward a specific commit or the remote repository.

**43. What is a tracking branch in Git?**

Tracking branches are the local branches that are connected to a remote branch and have a direct relationship with them, allowing you to push or pull changes to remote branches easily.

**44. What is a Git bundle?**

A Git bundle is a file containing the entire history of a Git repository. This includes all the branches, tags, and commits. You can create a git bundle using the following command:

git bundle create <bundle\_file> <refs>

The <bundle\_file> argument is the name of the file that the bundle will be saved to. The <refs> argument specifies which branches, tags, or commits are to be included in the bundle.

**45. What is the purpose of the git show command?**

git show command is used to show one or more objects. This can be blobs, trees, tags, and commits.

**46. What is the use of a Git clone?**

The Git clone command lets us copy the existing Git repository. If we want to get a copy of the central repository then the best way to do it is using ‘cloning’.

**47. What is a version control system? Mention its types.**

A version control system (VCS) is a software tool used to create different project versions and store them in a repository. All modifications to the code are recorded and tracked by the VCS.

**Types of version control systems:**

Local version control systems have a database that maintains all the file changes on disk under revision control in a special format.

Centralized version control systems contain one repository, and each user gets their own working copy.

Distributed version control systems contain multiple repositories, each accessible to separate users with their own working copy.

**48. What are the advantages of Git over SVN?**

Since Git is an open source version control system it lets us run multiple versions of our project so that it shows the changes that are made to the code over time and if needed we can keep track of the changes that we have made. This means that a large number of developers can make their own changes and upload those changes so that the changes can be attributed to the particular developers.

**49. What are some strategies for managing large binary files in Git?**

To manage large binary files in Git effectively, one can consider these strategies:

i) Git Large File Storage (LFS)

ii) Separate Large File Repository

iii) Selective Cloning and Fetching

iv) Binary File Compression

v) Optimize Storage Usage

vi) Use External Storage Services

vii) Archive Old Files

**50. What is the use of the git tag -a command?**

The git tag -a command is used to create an annotated tag in git. Annotated tags include the tagger name, email, and date, along with a tagging message.

Syntax: git tag -a <tagname> -m “<tagmessage>” [<commit>]

git tag is the base command that is used to create tags in git.

-a stands for “annotated” and instructs git to create an annotated tag.

tagname: This is the name that you want to give to your tag.

-m “message”: This is optional; you can include a message with “-m” to describe the tag.

**Advanced Git Interview Questions for Experienced**

**51. What is the function of ‘git config’?**

The ‘Git config’ is a great way to configure our options for the Git installation. Using this command, we can define the repository behavior, preferences and user information.

**52. What are the constituents of the commit object contain?**

the state of a project at a given point of time is contained in a set of files

Parent object commit references

A 40-character string that uniquely identifies the commit object called a SHAI name

**53. What is HEAD in Git, and how many HEADs can be created in a repository?**

The reference to a commit object is called the HEAD. Every repository has a ‘Master’ which is the default head. There can be multiple heads in a repository.

Git Head

Git Branching

**54. What does a git pull origin master do?**

Git pull origin master pulls the master branch into your current branch from the remote called origin. It only affects your current branch, not your local master branch.

**55. Throw some light on the git worktree.**

git-worktree is a command used to manage multiple working trees attached to the same repository.

**Commands:**

**add <path> [<commit-ish>]:** Create a worktree at <path> and checkout <commit-ish> into it.

**list**: List the details of each worktree.The main worktree is listed first, followed by each of the linked worktrees.

**move**: Move a worktree to a new location. The main worktree or linked worktrees containing submodules cannot be moved with this command.

**remove**: Remove a worktree. Only clean worktrees (no untracked files and modifications in tracked files) can be removed.

**repair**: Repair worktree administrative files, if possible, if they have become corrupted or outdated due to external factors.

**56. How would you remove all untracked files in Git?**

There are multiple ways to remove all untracked files in Git.

i) Discard all untracked files: git clean -f command will permanently remove all the untracked files from your working directory.

ii) Remove specific untracked files: Use git clean -n to list all the untracked files without removing them, then identify the specific files you want to remove. Now, use git rm <filename> to remove the individual untracked files.

**57. What is the difference between git log and reflog?**

Both git log and git reflog are used to keep track of the history of your repository, but they serve different purposes.

git log focuses on the public record of the commit history of the repository, whereas the git reflog focuses on private, local records of all references that have been updated in the repository.

**58. When would you use the git stash command?**

git stash temporarily shelves (or stashes) changes you’ve made to your working copy so you can work on something else, and then come back and work on them later. Below are some key scenarios when you might use it.

**59. What is the difference between git merge --no-ff and git merge --squash?**

git merge –no-ff will create a commit for the merge even if it could happen with fast forward, whereas git merge –squash will combine all the commits of the feature branch into one commit during the merge.

**60. Explain the use of git rebase --interactive. How can it be used to clean up a commit history?**

git rebase -i allows interactively rewriting a series of commits by changing commit order, squashing commits together, splitting commits, editing messages, removing commits, etc. This can clean up history by consolidating small commits, rewording messages, and pruning unnecessary changes.

Some ways git rebase –interactive can be used to clean up a commit history are as follows:

Squash multiple small or repetitive commits into a single, larger commit to make the history more concise and readable.

Edit/reword commit messages that are unclear, incorrectly worded, or have spelling/grammar issues to improve readability of history.

**61. How can you revert a git rebase?**

There are a few ways to revert or undo a git rebase:

**i) Reset to the original branch tip commit:**

git reflog

# find the commit hash before the rebase

git reset --hard <commit-hash>

The reflog shows your previous HEAD commits going back for a few months. Find the commit hash from before the rebase and reset back to it. This completely erases all commits made since the rebase.

**ii) New revert commit**

git revert -m 1 <rebase-head-commit>

This creates a new commit that undoes all the changes introduced in the rebase head commit. Preserves all previous commits.

**62. What is the significance of the git reflog command, and how can it be used to recover lost commits?**

Ans: The git reflog command is used to show a log of all the references recorded in the local repository by Git. This includes commits, resets, rebases, merges, etc. The main significance of git reflog is that it allows us to recover lost commits.

Git reflog can recover lost commits:

Git reflog shows when that branch tip commit was at HEAD when you accidentally deleted your feature branch.

**62. What are Git hooks? Give examples of how they can be used to enforce code quality checks before commits or pushes.**

Git hooks are scripts that run automatically when certain Git events occur, such as committing, pushing, merging, etc. They can be used to enforce project workflows and policies around the Git commit/push process.

**63. What is the purpose of git worktree, and how does it differ from using branches?**

The git worktree command allows you to have multiple working directories tied to the same repository. Each working tree acts like a complete, isolated checkout, useful for working on different branches or tasks in parallel.

With Git branches, you can switch context to different commits easily but have just a single view of files in the working directory.

**64. How do you define a ‘conflict’ in git?**

If we want to merge a commit there is a change in one place and the same change already exists then while merging the Git will not be able to predict which is the change that needs to be taken precedence.

**65. What is the regular way for branching in GIT?**

The best way to create a branch in GIT is to have one ‘main’ branch and then create another branch for implementing the changes that we want to make. This is extremely useful when there are a large number of developers working on a single project.

**66. What are submodules in Git, and how do they work? Provide an example of when you would use them.**

Git submodules allow you to embed external Git repositories inside a parent Git repository at a specific path. Some key properties of submodules:

Submodules clone a separate Git repo that has its own commit history, tags, etc. They act like nested repositories.

Parent repo records a fixed commit for the submodule. Updating the submodule commits requires an explicit update in parent.

Useful for modular components of a project that have separate release cycles or authors.

Changes in the submodule repo are not automatically reflected in the parent repo that uses it.

A good example using submodules:

A large project relies on a common shared library called common-utils.

The utils library has a separate dev team that manages releases at its own pace.

The main app can embed common-utils as a submodule placed inside src/utils.

When common-utils has a new feature or fix, the main app repo can periodically absorb those by updating the recorded submodule commit.  
  
Here, submodules allow modularizing a large codebase into standalone pieces that can evolve independently.

**Github Interview Questions**

**67. What is a clone on GitHub?**

Cloning in a git repository means you can create a local copy of the code provided by the developer. You can perform cloning with CLI.

**68. What do you know about GitHub and its repository?**

GitHub is a code-hosting platform for version control and collaboration. A repository is one of the elements of GitHub where you can store your code and files. A repository can have multiple collaborators and can be either public or private.

**69. Can you tell us a few benefits of using GitHub over other platforms?**

Used by more than 4 million organizations and with more than 100 million developers, GitHub is one of the most popular platforms. Below are some of the benefits of using GitHub:

i) Easy to use

ii) Robust documentation and support

iii) Collaboration

**70. How can users enhance the functionality of a branch in Git?**

Users can enhance the functionality of a branch in Git by adding the desired feature to any of the branches using the Git merge command. Since there is no limit on adding features to a branch, any branch can have any number of features.

**Git Commands Interview Questions**

**71. How do you initialize a new Git repository?**

To start keeping track of the changes, in your project using Git begin by opening the folder where your project is located on your computer. Next access the command line. Enter the command “git init”. This instructs Git to begin monitoring this folder. As a result Git will create a folder named “.git” within your project directory. This special folder stores data, about any modifications you make over time enabling Git to maintain a record of your projects history.

**72. What is the difference between “git fetch” and “git pull” ?**

When you use the “git fetch” command it downloads any changes, from the repository without merging them into your branch. This enables you to review the changes before merging them. On the hand when you use “git pull ” it not fetches changes from the remote repository but also automatically merges them into your current branch incorporating the latest remote changes immediately.

**73. What is a branch in Git and how do you create one?**

Git branches allow parallel development. Create a new branch with “git branch <name>” to build features or fix bugs separately. Or use “git checkout -b <name>” to make and switch to the new branch in one step, keeping that work isolated from the main codebase.

**74. Describe the process of resolving a merge conflict in Git.**

When you encounter a merge conflict you will need to make edits, to the files in order to resolve the conflicting changes. After that you should use “git add” to stage these changes and then commit them with “git commit“. This process allows you to carefully oversee how changes, from branches are integrated ensuring that your projects codebase remains intact.

**75. How do you revert a Git repository to a previous commit?**

To revert any changes you can make use of the “git reset” command. This command allows you to move the branch to a commit affecting both the staging area and working directory depending on the chosen mode (mixed or hard). Alternatively you can also utilize the “git revert” command which generates a commit, to changes made in a previous commit. This method is especially useful when dealing with changes that have already been pushed to shared repositories.

**Basic Questions**

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

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

**2. What is Git?**

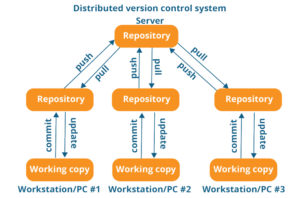
I will suggest you attempt this question by first telling about the architecture of git as shown in the below diagram just try to explain the diagram by saying:

* Git is a Distributed Version Control system(DVCS). It lets you track changes made to a file and allows you to revert back to any particular change that you wish.
* It is a distributed architecture that provides many advantages over other Version Control Systems (VCS) like SVN. One of the major advantages 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 available on his hard drive. So when there is a server outage all you need to do to recover is one of your teammate’s local Git repository.
* There is a central cloud repository where developers can commit changes and share them with other teammates.



**3. What is a distributed VCS?**

* These are the systems that don’t rely on a central server to store a project file and all its versions.
* In Distributed VCS, every contributor can get a local copy or “clone” of the main repository.
* As you can see in the above diagram, every programmer can maintain a local repository which is actually the copy or clone of the central repository which is present on their hard drive. They can commit and update their local repository without any hassles.
* With an operation called “pull”, they can update their local repositories with new data from the central server and “pull” operation affects changes to the main repository from their local repository.

****

**4. What is the difference between Git and Github?**

[Git](https://bit.ly/31MeW9b) is a version control system of distributed nature that is used to track changes in source code during software development. It aids in coordinating work among programmers, but it can be used to track changes in any set of files. The main objectives of Git are speed, data integrity, and support for distributed, non-linear workflows.

[GitHub](https://bit.ly/2rVhL7Q) is a Git repository hosting service, plus it adds many of its own features. GitHub provides a Web-based graphical interface. It also provides access control and several collaboration features, basic task management tools for every project.

**5. What are the benefits of using Version Control System?**

* With the Version Control System(VCS), all the team members are allowed to work freely on any file at any time. VCS gives you the flexibility to merge all the changes into a common version.
* All the previous versions and variants are neatly packed up inside the VCS. You can request any version at any time as per your requirement and you’ll have a snapshot of the complete project right at hand.
* Whenever you save a new version of your project, your VCS requires you to provide a short description of the changes that you have made. Additionally, you can see what changes are made in the file’s content. This helps you to know what changes have been made in the project and by whom.
* A distributed VCS like Git allows all the team members to have a complete history of the project so if there is a breakdown in the central server you can use any of your teammate’s local Git repository.

**6. 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.

**7. Mention the various Git repository hosting functions.**

* Github
* Gitlab
* Bitbucket
* SourceForge
* GitEnterprise

**8. What is a commit message?**

The 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.

**9. How can you fix a broken commit?**

In order to fix any broken commit, use the command “git commit --amend”. When you run this command, you can fix the broken commit message in the editor.

**10. What is a repository in Git?**

Repository in Git is a place where Git stores all the files. Git can store the files either on the local repository or on the remote repository.

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

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

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

**12. What is ‘bare repository’ in Git?**

A “bare” repository in Git contains information about the version control 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, whereas the working directory consists of :

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

**13. What is a ‘conflict’ in git?**

Git can handle on its own most merges by using its automatic merging features. There arises a conflict when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other. Conflicts are most likely to happen when working in a team environment.

**14. How is git instaweb used?**

‘git instaweb’ is used to automatically direct a web browser and run a webserver with an interface into your local repository.

**15. What is git is-tree?**

‘git is-tree’ represents a tree object including the mode and the name of each item and the SHA-1 value of the blob or the tree.

**16. Name a few Git commands and explain their usage.**

          Below are some basic Git commands:

|  |  |
| --- | --- |
| Command | Function |
| git rm [file] | deletes the file from your working directory and stages the deletion. |
| git log | list the version history for the current branch. |
| git show [commit] | shows the metadata and content changes of the specified commit. |
| git tag [commitID] | used to give tags to the specified commit. |
| git checkout [branch name]  git checkout -b [branch name] | used to switch from one branch to another.  creates a new branch and also switches to it. |

**Intermediate level Questions**

**17. How to resolve a conflict in Git?**

The following steps will resolve conflict in Git-

1. Identify the files that have caused the conflict.
2. Make the necessary changes in the files so that conflict does not arise again.
3. Add these files by the command git add.
4. Finally to commit the changed file using the command git commit

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

There can be two approaches to tackle 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 then push it to the remote repository. This is the most obvious way to fix an error. Once you have made necessary changes to the file, then commit it to the remote repository using the command: git commit -m “commit message”
* Also, you can create a new commit that undoes all changes that were made in the bad commit. To do this use the command

git revert <name of bad commit>

**19. What is SubGit?**

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

Now you can also 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 an existing Subversion repository. You can push to Git or commit to Subversion as per your convenience. Synchronization will be done by SubGit.

**20. 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

**21. What is ‘staging area’ or ‘index’ in Git?**

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.

**A diagram of a remote work flow

Description automatically generated**

**22. What work is restored when the deleted branch is recovered?**

The files which were stashed and saved in the stash index list will be recovered back. Any untracked files will be lost. Also, it is a good idea to always stage and commit your work or stash them.

If you want to fetch the log references of a particular branch or tag then run the command – “git reflog <ref\_name>”.

**23. What is 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 some time 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.

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.

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

If you want to continue working where you had left your work then ‘git stash apply‘ command is used to bring back the saved changes onto your current working directory.

**25. What is the difference between the ‘git diff ’and ‘git status’?**

‘git diff ’ depicts the changes between commits, commit and working tree, etc. whereas ‘git status’ shows you the difference between the working directory and the index, it is helpful in understanding a git more comprehensively. ‘git diff’ is similar to ‘git status’, the only difference is that it shows the differences between various commits and also between the working directory and index.

**26. What is the difference between ‘git remote’ and ‘git clone’?**

‘git remote add’ creates an entry in your git config that specifies a name for a particular URL whereas ‘git clone’ creates a new git repository by copying an existing one located at the URL

**27. What is 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}**.

**28. How do you find a list of files that have 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 file 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.

**29. What is the function of ‘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 a 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 a username.  
**git config –global user.email “Your E-mail Address”:**This command will add an email id.

**30. What does a commit object contain?**

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

* A set of files, representing the state of a project at a given point of time
* Reference to parent commit objects
* An SHA-1 name, a 40 character string that uniquely identifies the commit object

**31. Describe the branching strategies you have used.**

* **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 the develop branch, which may have progressed since the release was initiated.
* In the end tell them that branching strategies vary from one organization to another so I know basic branching operations like delete, merge, checking out a branch, etc.

**32. Explain the advantages of forking workflow**

* There is a fundamental difference between the forking workflow and other popular git workflows. Rather than using a single server-side to act as the “central” codebase, it gives every developer their own server-side repository. The Forking Workflow is commonly seen in public open-source projects.
* A crucial advantage of the Forking Workflow is that contributions can be integrated without even needing everybody to push to a single central repository that leads to clean project history. Developers can push to their own server-side repositories, but only the project maintainer can push to the official repository.
* If developers are ready to publish a local commit, then they push the commit to their own public repository and not the official one. After this, they go for a pull request with the main repository that lets the project maintainer know an update is ready to be integrated.

**33. 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.

**34. Why is it desirable to create an additional commit rather than amending an existing commit?**

There are a couple of reasons for this –

1. The amend operation destroys the state that was previously saved in a commit. If there is just the commit message being changed then that’s not a problem.  But if the contents are being amended then chances of eliminating something important remains more.
2. Abusing “git commit- amend” can result in the growth of a small commit and acquire unrelated changes.

**35. What does ‘hooks’ comprise of in Git?**

This directory consists of shell scripts that are activated if you run the corresponding Git commands.  For example, git will try to execute the post-commit script after you have run a commit.

**36. In Git, how would you return a commit that has just been pushed and made open?**

One or more commits can be reverted through the use of git revert. This command, in a true sense, creates a new commit with patches that cancel out the changes introduced in specific commits. If in case the commit that needs to be reverted has already been published or changing the repository history is not an option then in such cases, git revert can be used to revert commits. If you run the following command then it will revert the last two commits:

git revert HEAD~2..HEAD

Alternatively, there is always an option to check out the state of a particular commit from the past and commit it anew.

**37. How to remove a file from git without removing it from your file system?**

One has to be careful during a git add, else you may end up adding files that you didn’t want to commit. However, git rm will remove it from both your staging area (index), as well as your file system (working tree), which may not be what you want.

Instead, use git reset:

git reset filename          # or

echo filename >> .gitingore # add it to .gitignore to avoid re-adding it

This means that git reset <paths> is exactly the opposite of git add <paths>.

**38. Can you explain the Gitflow workflow?**

To record the history of the project, Gitflow workflow employs two parallel long-running branches – master and develop:

* Master – this branch is always ready to be released on LIVE, with everything fully tested and approved (production-ready).
* Hotfix – these branches are used to quickly patch production releases. These branches are a lot like release branches and feature branches except they’re based on master instead of develop.
* Develop – this is the branch to which all feature branches are merged and where all tests are performed. Only when everything’s been thoroughly checked and fixed it can be merged to the master.
* Feature – each new feature should reside in its own branch, which can be pushed to the develop branch as their parent one.

**39. Tell me the difference between HEAD, working tree and index, in Git.**

* The working tree/working directory/workspace is the directory tree of (source) files that you are able to see and edit.
* The index/staging area is a single, large, binary file in <baseOfRepo>/.git/index, which lists all files in the current branch, their SHA-1 checksums, timestamps, and the file name – it is not another directory which contains a copy of files in it.
* HEAD is used to refer to the last commit in the currently checked-out branch.

**40. What is Git fork? What is the difference between fork, branch, and clone?**

* A fork is a copy of a repository. Normally you fork a repository so that you are able to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else’s project or to use someone else’s project as a starting point for your own idea.
* git cloning means pointing to an existing repository and make a copy of that repository in a new directory, at some other location. The original repository can be located on the local file system or on remote machine accessible supported protocols. The git clone command is used to create a copy of an existing Git repository.
* In very simple words, git branches are individual projects within a git repository. Different branches within a repository can have completely different files and folders, or it could have everything the same except for some lines of code in a file.

**41. What are the different ways you can refer to a commit?**

* In Git each commit has a unique hash. These hashes are used to identify the corresponding commits in various scenarios, for example, while trying to checkout a particular state of the code using the git checkout {hash} command.
* Along with this, Git maintains a number of aliases to certain commits, known as refs. Also, every tag that is created in the repository effectively becomes a ref and that is exactly why you can use tags instead of committing hashes in various git commands. Git also maintains a number of special aliases that are changed based on the state of the repository, such as HEAD, FETCH\_HEAD, MERGE\_HEAD, etc.
* In Git, commits are allowed to be referred to as relative to one another. In the case of merge commits, where the commit has two parents, ^ can be used to select one of the two parents, for example, HEAD^2 can be used to follow the second parent.
* And finally, refspecs are used to map local and remote branches together. However, these can also be used to refer to commits that reside on remote branches allowing one to control and manipulate them from a local git environment.

**42. What is the difference between rebasing and merge in Git?**

* In Git, the rebase command is used to integrate changes from one branch into another. It is an alternative to the “merge” command. The difference between rebasing and merge is that rebase rewrites the commit history in order to produce a straight, linear succession of commits.
* Merging is Git’s way of putting a forked history back together again. The git merge command helps you take the independent lines of development created by git branch and integrate them into a single branch.

**43. Explain the difference between reverting and resetting.**

* Git reset is a powerful command that is used to undo local changes to the state of a Git repository. Git reset operates on “The Three Trees of Git” which are, Commit History ( HEAD ), the Staging Index, and the Working Directory.
* Revert command in Git creates a new commit that undoes the changes from the previous commit. This command adds a new history to the project. It does not modify the existing history.

**44. What is git cherry-pick?**

The command git cherry-pick is normally used to introduce particular commits from one branch within a repository onto a different branch. Another common use is to forward- or back-port commits from a maintenance branch to a development branch. This is in contrast with other ways such as merge and rebase which normally apply many commits onto another branch.

Consider:

git cherry-pick <commit-hash>

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

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.

The output will also include some extra information, which can be easily suppressed by including a couple of 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.

**Advanced level Questions**

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

**There are two options to squash the 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)**)”**

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

* Git bisect is used to find the commit that introduced a bug by using binary search. The command for Git bisect is  
  **git bisect <subcommand> <options>**
* Now since you have mentioned the command above explain to 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.

**48. 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.

Sanity or smoke testdetermines 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.

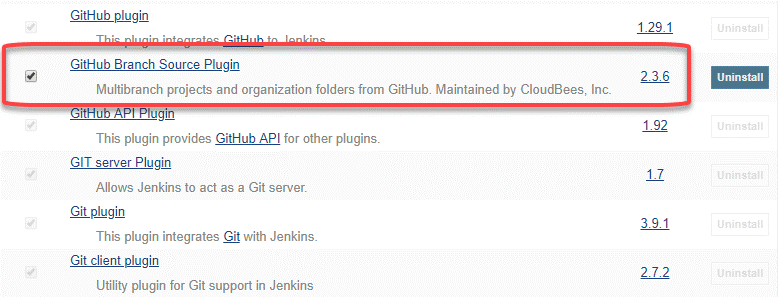
**49. How do you integrate Git with Jenkins?**

**Step 1**. Click on the manage jenkins button on your jenkins dashboard.

**Step 2**. Click on manage jenkins plugin.

**Step 3:**In the Plugins Page

1. Select the GIT Plugin
2. Click on **Install without restart.**The plugin will take a few moments to finish downloading depending on your internet connection, and will be installed automatically.
3. You can also select the option **Download now and Install after restart** In which plugin is installed after restart
4. You will be shown a “No updates available” message if you already have the Git plugin installed.

**Step 4**: Once the plugins have been installed, go to **Manage Jenkins** on your Jenkins dashboard. You will see your plugins listed among the rest.

**50. What is git reflog?**

The ‘reflog’ command keeps a **track of** **every single change made in the references**(branches or tags) of a repository and keeps a log history of the branches and tags that were either created locally or checked out. Reference logs such as the commit snapshot of when the branch was created or cloned, checked-out, renamed, or any commits made on the branch are maintained by [Git](https://www.edureka.co/blog/what-is-git/) and listed by the ‘reflog’ command.

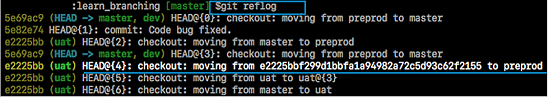
***Note:****The branch will be recoverable from your working directory only if the branch ever existed in your local repository i.e. the branch was either created locally or checked-out from a remote repository in your local repository for Git to store its reference history logs.*

This command must be executed in the repository that had the lost branch. If you consider the remote repository situation, then you have to execute the reflog command on the developer’s machine who had the branch.

***command:****git reflog*

**51. How to recover a deleted branch using git reflog?**

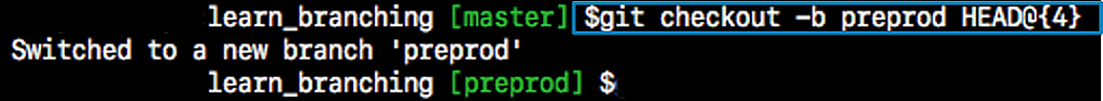
**Step 1**: **History logs of all the references**

Get a list of all the local recorded history logs for all the references (‘master’, ‘uat’ and ‘prepod’) in this repository.

**Step 2: Identify the history stamp**

As you can see from the above snapshot, the highlighted commit id: e2225bb along with the HEAD pointer index:4 is the one when ‘preprod’ branch was created from the current HEAD pointer pointing to your latest work.

**Step 3**: **Recover**

If you want to recover back the ‘preprod‘ branch then use the command  ‘git checkout’ passing the HEAD pointer reference with the index id – 4. This is the pointer reference when ‘preprod’ branch was created long commit id highlighted in the output screenshot.

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

git fetch only downloads new data from a remote repository, but it doesn’t integrate any of the downloaded data into your working files. All it does is provide a view of this data.

git pull downloads as well as merges the data from a remote repository into your local working files. It may also lead to merge conflicts if your local changes are not yet committed. Use the git stash command to hide your local changes.

**2. How do you revert a commit that has already been pushed and made public?**

One or more commits can be reverted through the use of *git revert*. This command, in essence, 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

Alternatively, one can always checkout the state of a particular commit from the past, and commit it anew.

**3. 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.

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

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.

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

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

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

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

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.

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

Git provides a rather efficient mechanism to find bad commits. Instead of making the user try out every single commit to find out the first one that introduced some particular issue into the code, *git bisect* allows the user to perform a sort of binary search on the entire history of a repository.

By issuing the command *git bisect start*, the repository enters bisect mode. After this, all you have to do is identify a bad and a good commit:

git bisect bad # marks the current version as bad

git bisect good {hash or tag} # marks the given hash or tag as good, ideally of some earlier commit

Once this is done, Git will then have a range of commits that it needs to explore. At every step, it will checkout a certain commit from this range, and require you to identify it as good or bad. After which the range will be effectively halved, and the whole search will require a lot less number of steps than the actual number of commits involved in the range. Once the first bad commit has been found, or the bisect mode needs to be ended, the following command can be used to exit the mode and reset the bisection state:

git bisect reset

**7.What are the different ways you can refer to a commit?**

In Git each commit is given a unique hash. These hashes can be used to identify the corresponding commits in various scenarios (such as while trying to checkout a particular state of the code using the *git checkout {hash}* command).

Additionally, Git also maintains a number of aliases to certain commits, known as refs. Also, every tag that you create in the repository effectively becomes a ref (and that is exactly why you can use tags instead of commit hashes in various git commands). Git also maintains a number of special aliases that change based on the state of the repository, such as HEAD, FETCH\_HEAD, MERGE\_HEAD, etc.

Git also allows commits to be referred as relative to one another. For example, HEAD~1 refers to the commit parent to HEAD, HEAD~2 refers to the grandparent of HEAD, and so on. In case of merge commits, where the commit has two parents, ^ can be used to select one of the two parents, e.g. HEAD^2 can be used to follow the second parent.

And finally, refspecs. These are used to map local and remote branches together. However, these can be used to refer to commits that reside on remote branches allowing one to control and manipulate them from a local Git environment.

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

Hide answer

In simple words, *git rebase* allows one to move the first commit of a branch to a new starting location. For example, 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.

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

Hide answer

This can be done with a simple script bound 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. For example, the following 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

… checks to see if any .go file that is about to be commited 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.

**10.One of your teammates accidentally deleted a branch, and has already pushed the changes to the central git repo. There are no other git repos, and none of your other teammates had a local copy. How would you recover this branch?**

Check out the latest commit to this branch in the reflog, and then check it out as a new branch.

**11. How can you copy a commit made in one branch to another (e.g. a hot fix commit from released branch to current development branch)?**

Hide answer

You need to use the cherry-pick command. It provides the possibility to play back an existing commit to your current location/branch. So you need to switch to the target branch (e.g. git checkout development) and call git cherry-pick {hash of that commit}.

In spite of applying the same changes, it will be a new commit with a new hash because the changes are applied to a different destination.

**12.How do you cherry-pick a merge commit?**

Cherry-pick uses a diff to find the difference between branches.

As a merge commit belongs to a different branch, it has two parents and two changesets.

For example, if you have merge commt ref 63ad84c, you have to specify -m and use parent 1 as a base:

git checkout release\_branch

git cherry-pick -m 1 63ad84c

**13.What is a conflict in git and how can it be resolved?**

A conflict arises when more than one commit that has to be merged has some change in the same place or same line of code. Git will not be able to predict which change should take precedence. This is a git conflict.

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 merge, so it sets the parents of the commit correctly.

**GIT Interview Questions javaTpoint.**

**1) What is GIT?**

Git is an open source distributed version control system and source code management (SCM) system with an insistence to control small and large projects with speed and efficiency.

**2) Which language is used in Git?**

Git uses 'C' language. Git is quick, and 'C' language makes this possible by decreasing the overhead of run times contained with high-level languages.

**3) What is a repository in Git?**

A repository consists of a list named .git, where git holds all of its metadata for the catalog. The content of the .git file is private to Git.

**4) What is 'bare repository' in Git?**

A "bare" repository in Git includes the version control information and no working files (no tree), and it doesn?t include the special. git sub-directory. Instead, it consists of all the contents of the .git sub-directory directly in the main directory itself, whereas working list comprises of:

Backward Skip 10sPlay VideoForward Skip 10s

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

**5) What is the purpose of GIT stash?**

GIT stash takes the present state of the working file and index and puts in on the stack for next and gives you back a clean working file. So in case if you are in the middle of object and require to jump over to the other task, and at the same time you don't want to lose your current edits, you can use GIT stash.

**6) What is GIT stash drop?**

When you are done with the stashed element or want to delete it from the directory, run the git 'stash drop' command. It will delete the last added stash item by default, and it can also remove a specific topic if you include as an argument.

**7) What are the advantages of using GIT?**

Here are some of the essential advantages of Git:

* Data repetition and data replication is possible
* It is a much applicable service
* For one depository you can have only one directory of Git
* The network performance and disk application are excellent
* It is effortless to collaborate on any project
* You can work on any plan within the Git

**8) What is the function of 'GIT PUSH' in GIT?**

'GIT PUSH' updates remote refs along with related objects

**9) Why do we require branching in GIT?**

With the help of branching, you can keep your branch, and you can also jump between the different branches. You can go to your past work while at the same time keeping your recent work intact.

**10) What is the purpose of 'git config'?**

The 'Git config' is a great method to configure your choice for the Git installation. Using this command, you can describe the repository behavior, preferences, and user information.

**11) What is the definition of "Index" or "Staging Area" in GIT?**

When you are making the commits, you can make innovation to it, format it and review it in the common area known as 'Staging Area' or 'Index'.

**12) What is a 'conflict' in git?**

A 'conflict' appears when the commit that has to be combined has some change in one place, and the current act also has a change at the same place. Git will not be easy to predict which change should take precedence.

**13) What is the difference between git pull and git fetch?**

Git pull command pulls innovation or commits from a specific branch from your central repository and updates your object branch in your local repository.

Git fetch is also used for the same objective, but it works in a slightly different method. When you behave a git fetch, it pulls all new commits from the desired branch and saves it in a new branch in your local repository. If you need to reflect these changes in your target branch, git fetch should be followed with a git merge. Your target branch will only be restored after combining the target branch and fetched branch. To make it simple for you, remember the equation below:

**Git pull = git fetch + git merge**

**14) How to resolve a conflict in Git?**

If you need to resolve a conflict in Git, edit the list for fixing the different changes, and then you can run "git add" to add the resolved directory, and after that, you can run the 'git commit' for committing the repaired merge.

**15) What is the purpose of the git clone?**

The git clone command generates a copy of a current Git repository. To get the copy of a central repository, 'cloning' is the simplest way used by programmers.

**16) What is git pull origin?**

pull is a get and a consolidation. 'git pull origin master' brings submits from the master branch of the source remote (into the local origin/master branch), and then it combines origin/master into the branch you currently have looked out.

**17) What does git commit a?**

Git commits "records changes to the storehouse" while git push " updates remote refs along with contained objects" So the first one is used in a network with your local repository, while the latter one is used to communicate with a remote repository.

**18) Why GIT better than Subversion?**

GIT is an open source version control framework; it will enable you to run 'adaptations' of a task, which demonstrate the changes that were made to the code over time also it allows you keep the backtrack if vital and fix those changes. Multiple developers can check out, and transfer changes, and each change can then be attributed to a particular developer.

**19) Explain what is commit message?**

Commit message is a component of git which shows up when you submit a change. Git gives you a content tool where you can enter the adjustments made to a commit.

**20) Why is it desirable to create an additional commit rather than amending an existing commit?**

There are couples of reason

1. The correct activity will devastate the express that was recently saved in a commit. If only the commit message gets changed, that's not a problem. But if the contents are being modified, chances of excluding something important remains more.
2. Abusing "git commit- amends" can cause a small commit to increase and acquire inappropriate changes.

**21) What does 'hooks' comprise of in Git?**

This index comprises of Shell contents which are enacted after running the relating git commands. For instance, Git will attempt to execute the post-commit content after you run a commit.

**22) What is the distinction between Git and Github?**

A) Git is a correction control framework, a tool to deal with your source code history.

GitHub is a hosting function for Git storehouses.

GitHub is a website where you can transfer a duplicate of your Git archive. It is a Git repository hosting service, which offers the majority of the distributed update control and source code management (SCM) usefulness of Git just as including its features.

**23) In Git, how would you return a commit that has just been pushed and made open?**

There can be two answers to this question and ensure that you incorporate both because any of the below choices can be utilized relying upon the circumstance:

Remove or fix the bad document in another commit and push it to the remote repository. This is a unique approach to correct a mistake. Once you have necessary changes to the record, commit it to the remote repository for that I will utilize

**git submit - m "commit message."**

Make another commit that fixes all changes that were made in the terrible commit. to do this, I will utilize a command

**git revert <name of bad commit>**

**24) What does the committed item contain?**

Commit item contains the following parts; you should specify all the three present below:

A set of records, representing to the condition of a task at a given purpose of time

References to parent commit objects

An SHAI name, a 40 character string that uniquely distinguishes the commit object.

**25) Describing branching systems you have utilized?**

This question is a challenge to test your branching knowledge with Git along these lines, inform them regarding how you have utilized branching in your past activity and what reason does it serves, you can refer the below mention points:

**Feature Branching:**

A component branch model keeps the majority of the changes for a specific element within a branch. At the point when the item is throughout tested and approved by automated tests, the branch is then converged into master.

**Task Branching**

In this model, each assignment is actualized on its branch with the undertaking key included in the branch name. It is anything but difficult to see which code actualizes which task, search for the task key in the branch name.

**Release Branching**

Once the create branch has procured enough features for a discharge, you can clone that branch to frame a Release branch. Making this branch begins the following discharge cycle so that no new features can be included after this point, just bug fixes, documentation age, and other release oriented assignments ought to go in this branch. When it is prepared to deliver, the release gets converged into master and labeled with a form number. Likewise, it should be converged once again into creating a branch, which may have advanced since the release was started.

At last, disclose to them that branching methodologies fluctuate starting with one association then onto the next, so I realize essential branching activities like delete, merge, checking out a branch, etc.

**26) By what method will you know in Git if a branch has just been combined into master?**

The appropriate response is immediate.

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

**git branch - merged** It records the branches that have been merged into the present branch.

**git branch - no merged** It records the branches that have not been merged.

**27) How might you fix a messed up submit?**

To fix any messed up commit, you will utilize the order "git commit?correct." By running this direction, you can set the wrecked commit message in the editor.

**28) Mention the various Git repository hosting functions.**

The following are the Git repository hosting functions:

* Pikacode
* Visual Studio Online
* GitHub
* GitEnterprise
* SourceForge.net

**29) Mention some of the best graphical GIT customers for LINUX?**

Some of the best GIT customer for LINUX is

1. Git Cola
2. Smart git
3. Git-g
4. Git GUI
5. Giggle
6. qGit

**30) What is Subgit? Why use it?**

'Subgit' is a tool that migrates SVN to Git. It is a stable and stress-free migration. Subgit is one of the solutions for a company-wide migration from SVN to Git that is:

1. It is much superior to git-svn
2. No need to change the infrastructure that is already placed.
3. It allows using all git and all sub-version features.
4. It provides stress ?free migration experience.