**Basic Git Interview Questions:**

1. **What is Git?**
   * Git is a version control system for tracking changes in computer files and is used to help coordinate work among several people on a project while tracking progress over time. In other words, it’s a tool that facilitates source code management in software development.
2. **What do you understand by the term ‘Version Control System’?**
   * A version control system (VCS) records all the changes made to a file or set of data, so a specific version may be called later if needed.
3. **What is GitHub?**
   * To provide Internet hosting for version control and software development, GitHub makes use of Git.
4. **Mention some popular Git hosting services.**
   * GitHub, SourceForge, GitLab, Bitbucket
5. **Different types of version control systems.**
   * Local version control systems, Centralized version control systems, Distributed version control systems
6. **What benefits come with using GIT?**
   * Data replication and redundancy are both possible. It is a service with high availability. There can only be one Git directory per repository. Excellent network and disc performance are achieved. On any project, collaboration is straightforward.
7. **What’s the difference between Git and GitHub?**
   * **Git:** Software, Local installation, Git GUI interface
   * **GitHub:** Service, Hosted on the web, GitHub Desktop interface, Built-in user management
8. **What is a Git repository?**
   * Git repository refers to a place where all the Git files are stored. These files can either be stored on the local repository or on the remote repository.
9. **How can you initialize a repository in Git?**
   * If you want to initialize an empty repository to a directory in Git, you need to enter the git init command. After this command, a hidden .git folder will appear.
10. **How is Git different from Subversion (SVN)?**
    * **GIT:** Distributed decentralized version control system, Stores content in the form of metadata, The master contains the latest stable release, Contents hashed using SHA-1
    * **SVN:** Centralized version control system, Stores data in the form of files, Trunk directory has the latest stable release, Doesn’t support hashed contents
11. **Name a few Git commands with their function.**
    * Git config - Configure the username and email address
    * Git add - Add one or more files to the staging area
    * Git diff - View the changes made to the file
    * Git init - Initialize an empty Git repository
    * Git commit - Commit changes to head but not to the remote repository
12. **What are the advantages of using Git?**
    * Faster release cycles, Easy team collaboration, Widespread acceptance, Maintains the integrity of source code, Pull requests
13. **What language is used in Git?**
    * Git is a fast and reliable version control system, and the language that makes this possible is ‘C.’
14. **What is the correct syntax to add a message to a commit?**
    * **git commit -m "x files created"**
15. **Which command is used to create an empty Git repository?**
    * **git init** - This command helps to create an empty repository while working on a project.
16. **What does git pull origin master do?**
    * The **git pull origin master** fetches all the changes from the master branch onto the origin and integrates them into the local branch.

**Intermediate Git Interview Questions:**

1. **What does the git push command do?**
   * The Git push command is used to push the content in a local repository to a remote repository. After a local repository has been modified, a push is executed to share the modifications with remote team members.
2. **Difference between git fetch and git pull.**
   * **Git Fetch:** Only downloads new data from a remote repository. Does not integrate any of these new data into your working files. **git fetch origin**
   * **Git Pull:** Updates the current HEAD branch with the latest changes from the remote server. Downloads new data and integrates it with the current working files. **git pull origin master**
3. **GitHub, GitLab, and Bitbucket are examples of git repository \_\_\_\_\_\_\_ function?**
   * Hosting. All the three are services for hosting Git repositories.
4. **What do you understand about the Git merge conflict?**
   * A Git merge conflict is an event that occurs when Git is unable to resolve the differences in code between the two commits automatically. Git is capable of automatically merging the changes only if the commits are on different lines or branches.
5. **How do you resolve conflicts in Git?**
   * Identify the files responsible for the conflicts.
   * Implement the desired changes to the files.
   * Add the files using the **git add** command.
   * Commit the changes in the file with the help of the **git commit** command.
6. **What is the functionality of git ls-tree?**
   * The **git ls-tree** command is used to list the contents of a tree object.
7. **What is the process to revert a commit that has already been pushed and made public?**
   * Two processes:
     1. Remove or fix the bad file in a new commit and push it to the remote repository. Commit it to the remote repository using: **git commit –m “commit message”**
     2. Create a new commit to undo all the changes made in the bad commit. Use the following command: **git revert <commit id>**
8. **How is a bare repository different from the standard way of initializing a Git repository?**
   * **Standard way:** Create a working directory with the **git init** command, A **.git** subfolder is created with all the git-related change history.
   * **Bare way:** Does not contain any working or checked out copy of source files, Bare repositories store git revision history in the root folder of your repository instead of the **.git** subfolder.
9. **What does git clone do?**
   * Git clone allows you to create a local copy of the remote GitHub repository. Once you clone a repo, you can make edits locally in your system rather than directly in the source files of the remote repo.
10. **What is Git stash?**
    * Git stash takes your modified tracked files and saves them on a stack of unfinished changes that you can reapply at any time. It is used when you want to switch branches to work on something else without making commits in uncompleted work.
11. **What does the git reset --mixed and git merge --abort commands do?**
    * **git reset --mixed** is used to undo the last commit while keeping changes staged. It resets the index but not the working directory.
    * **git merge --abort** is used to abort the merge process, discarding all changes.
12. **How do you delete a branch in Git?**
    * To delete a branch in Git, you can use the command **git branch -d <branch\_name>**.
13. **How to rename a Git branch?**
    * To rename a branch in Git, you can use the command **git branch -m <new\_branch\_name>**.
14. **What is Git rebase?**
    * Git rebase is a command used to integrate changes from one branch into another. The primary advantage of rebasing is that it allows you to keep a linear project history.
15. **Explain the git cherry-pick command.**
    * The **git cherry-pick** command is used to apply the changes introduced by some existing commits. It allows you to pick a specific commit from one branch and apply it onto another branch.
16. **What is the purpose of .gitignore in a Git repository?**
    * The **.gitignore** file specifies intentionally untracked files that Git should ignore. It is useful to avoid committing unnecessary files and directories, such as temporary files and build artifacts, into the version control system.
17. **How do you check the differences between the working directory and the staging area in Git?**
    * The **git diff** command can be used to check the differences between the working directory and the staging area in Git.
18. **What is the use of Git bisect?**
    * Git bisect is a command that uses binary search to help find the commit that introduced a bug. It is a powerful tool for identifying the commit that introduced a regression in the code.
19. **How can you rewrite Git commit history?**
    * Git commit history can be rewritten using commands like **git rebase** or **git commit --amend**. These commands allow you to modify and organize commits before pushing them to a remote repository.
20. **What is the purpose of Git hooks?**
    * Git hooks are scripts that Git executes before or after specific events such as commit, push, and receive. They are useful for implementing custom workflows and ensuring code quality and consistency.
21. **What is the difference between git rebase and git merge?**
    * **Git Rebase:** Integrates changes from one branch into another by moving, combining, or skipping commits. Results in a linear project history.
    * **Git Merge:** Integrates changes from one branch into another by creating a merge commit. Results in a more complex project history.
22. **How can you move or rename a file in Git?**
    * You can use the **git mv** command to move or rename a file in Git. It is equivalent to moving the file manually and then using **git add** on the new file and **git rm** on the old file.
23. **How does Git handle branching and merging?**
    * Git handles branching and merging through lightweight branches. Each branch is a simple file that contains a 40-character SHA-1 checksum of the commit it points to. Merging is the process of combining changes from different branches.
24. **What is a detached HEAD in Git?**
    * A detached HEAD state in Git occurs when you check out a commit directly, rather than a branch. This means you are no longer on any branch, and any commits you create in this state will be unreachable unless you create a branch to access them.
25. **How does Git store data?**
    * Git stores data as snapshots of a mini filesystem, where each commit is a full snapshot of the source code at a given point in time. It also uses a system called object storage to keep track of changes and files.
26. **What is the role of git reflog?**
    * **git reflog** is a reference log that helps you manage and restore local branches that may have been lost. It records changes in the state of your branch references, providing a safety net in case you accidentally delete or modify branches.
27. **How do you stop tracking a file in Git without deleting it?**
    * To stop tracking a file without deleting it, you can use the **git rm --cached** command. This removes the file from the staging area but preserves it in the working directory.
28. **What is the use of the git clean command?**
    * The **git clean** command is used to remove untracked files from your working directory. It is helpful when you want to clean up your local workspace by removing files that are not part of the Git repository.
29. **How do you view the commit history in Git?**
    * The **git log** command is used to view the commit history in Git. It displays a list of commits along with their details, including author, date, and commit message.
30. **What is the purpose of the git tag command?**
    * The **git tag** command is used to create, list, delete, or verify tags in the Git repository. Tags are references to specific points in Git history, often used to mark release points.
31. **How can you recover a deleted branch in Git?**
    * To recover a deleted branch in Git, you can use the **git reflog** to find the commit where the branch was deleted and then recreate the branch at that commit using the branch command.
32. **What is the significance of the .gitattributes file?**
    * The **.gitattributes** file allows you to specify attributes for pathnames in your repository. It is used to control the normalization of line endings, merge strategies for specific files, and other attributes.
33. **What is Git LFS?**
    * Git LFS (Large File Storage) is an extension for Git that handles large files by replacing them with text pointers in your Git repository while storing the actual file contents on a separate server.
34. **How do you set up and use Git LFS?**
    * To set up Git LFS, you need to install it using **git lfs install** and then track large files using **git lfs track <pattern>**. Finally, you need to commit and push the **.gitattributes** file and large files using regular Git commands.

**Advanced Git Interview Questions:**

1. **Explain the three main states in Git.**
   * **Working Directory:** The actual files reside here.
   * **Index (Staging Area):** AKA Cache, Files and changes are ready to be committed here.
   * **HEAD:** Points to the most recent commit.
2. **How does Git handle binary files?**
   * Git treats binary files as blobs and does not attempt to merge changes in binary files. It only stores the entire file and is not efficient in handling the history of binary files.
3. **What is a Git submodule?**
   * A Git submodule is a Git repository embedded inside another Git repository. It allows you to include or embed one Git repository within another, providing a way to manage dependencies.
4. **How does Git store passwords?**
   * Git can store passwords using credential helpers, which are external tools that securely store login information and provide it to Git on request. Examples include the Git Credential Manager and cache-based credential storage.
5. **Explain the difference between Git pull and Git fetch.**
   * **Git Pull:** Fetches the changes from a remote repository and merges them into the current branch. It's equivalent to running **git fetch** followed by **git merge**.
   * **Git Fetch:** Fetches the changes from a remote repository but does not automatically merge them. It updates the remote-tracking branches.
6. **What is the role of Git garbage collection?**
   * Git garbage collection is the process of cleaning up unnecessary files and optimizing the local repository. It helps in reclaiming space and improving the performance of the repository.
7. **How does Git handle line endings?**
   * Git has the ability to automatically convert line endings between the operating systems (Windows, Linux, macOS) using the **core.autocrlf** configuration. It ensures consistent line endings across different platforms.
8. **What is the purpose of the Git index?**
   * The Git index, also known as the staging area, is a crucial component in Git. It is a binary file that stores information about what will go into the next commit. When you perform a git add, you are updating the index.
9. **How do you split a commit in Git?**
   * To split a commit in Git, you can use an interactive rebase (**git rebase -i**) and choose to edit the commit. Then, use the **git reset** command to unstage changes, make the necessary modifications, and commit the changes separately.
10. **What is Git grafts and replace?**
    * **Git Grafts:** An older method used to combine histories of multiple projects. It involves manually specifying parent-child relationships between commits.
    * **Git Replace:** A more modern alternative, allowing you to replace one object with another. It is useful for grafting histories together.
11. **How does Git handle symbolic links?**
    * Git supports symbolic links, and it can either store them as pointers (on Windows) or as actual symbolic links (on Unix-like systems) in the repository. The behavior depends on the platform and Git settings.
12. **What is the purpose of the .gitkeep file?**
    * The **.gitkeep** file is a convention used to include an otherwise empty directory in the version control system. Git ignores empty directories, so adding a **.gitkeep** file helps in keeping the directory tracked.
13. **How does Git handle concurrent changes?**
    * Git uses a merge algorithm to automatically merge changes when different branches are merged. In case of conflicts, Git marks the conflicts, and it is up to the user to resolve them manually.
14. **How can you squash commits in Git?**
    * To squash commits in Git, you can use an interactive rebase (**git rebase -i**) and mark the commits as "squash" or "fixup." This allows you to combine multiple commits into a single commit.
15. **Explain Git subversion and its differences.**
    * Git Subversion (git-svn) is a bidirectional bridge that allows Git and Subversion repositories to interact. Developers can use Git locally and interact with a Subversion repository, enabling a gradual transition to Git.
16. **How do you cherry-pick a range of commits in Git?**
    * To cherry-pick a range of commits, you can use the **git cherry-pick <start-commit>^..<end-commit>** command. The **^** is used to exclude the starting commit itself.
17. **How does Git merge conflicts in binary files?**
    * Git does not attempt to automatically merge changes in binary files. In case of conflicts, Git marks the file as conflicted, and it's up to the user to manually resolve the conflict by selecting the appropriate version.
18. **What is the difference between git revert and git reset?**
    * **Git Revert:** Introduces a new commit that undoes the changes of a previous commit. Maintains a linear project history.
    * **Git Reset:** Moves the branch pointer to a specified commit, discarding all commits after that point. It can be used to rewrite history.
19. **Explain shallow cloning in Git.**
    * Shallow cloning in Git involves only fetching a limited history of a repository, reducing the download size. It can be done using the **--depth** option with the **git clone** command.
20. **How can you amend the last commit message in Git?**
    * To amend the last commit message in Git, you can use the **git commit --amend** command. This opens the default text editor, allowing you to modify the commit message.
21. **How does Git handle line-ending differences between Windows and Unix systems?**
    * Git can automatically convert line endings between Windows (CRLF) and Unix (LF) using the **core.autocrlf** configuration. This ensures consistent line endings when collaborating across different platforms.
22. **Explain what git fsck is used for.**
    * **git fsck** is used to perform a file system check on the Git repository. It verifies the connectivity and integrity of the objects in the repository and reports any issues or inconsistencies.
23. **What is Git rerere, and how does it work?**
    * **git rerere** stands for "reuse recorded resolution" and is a feature in Git that allows you to record conflict resolutions during merges and reuse them automatically in future merges. It helps in saving time when resolving similar conflicts.
24. **How can you undo the last commit but keep the changes in your working directory?**
    * To undo the last commit but keep the changes in the working directory, you can use the **git reset --soft HEAD^** command. This moves the branch pointer back by one commit while keeping the changes staged.
25. **What is the role of .gitignore in a Git repository?**
    * The **.gitignore** file specifies intentionally untracked files that Git should ignore. It helps in avoiding the accidental inclusion of unnecessary files in the version control system.
26. **How do you create a new branch in Git?**
    * To create a new branch in Git, you can use the **git branch <branch-name>** command. This creates a new branch without switching to it. To switch to the new branch, you can use **git checkout <branch-name>** or **git switch <branch-name>**.
27. **What is Git bisect, and how does it work?**
    * Git bisect is a command that helps you find the commit that introduced a bug. It uses a binary search algorithm, automatically checking out commits and allowing you to mark them as good or bad until the bug's origin is identified.
28. **How do you untrack a file in Git without deleting it?**
    * To untrack a file in Git without deleting it, you can use the **git rm --cached <file>** command. This removes the file from the staging area while keeping it in the working directory.
29. **How does Git calculate the SHA-1 checksum of a commit?**
    * Git calculates the SHA-1 checksum of a commit by considering the commit's content, parent commit(s), timestamp, author, and other metadata. Any change in these properties results in a different SHA-1 checksum.
30. **Explain what a Git commit object contains.**
    * A Git commit object contains metadata such as the author, committer, commit message, a reference to a tree object (representing the state of the project), and references to parent commit(s) if it is not the initial commit.
31. **What is Git cherry-pick, and when would you use it?**
    * Git cherry-pick is a command used to apply the changes introduced by some existing commits to another branch. It is useful when you want to selectively pick specific commits from one branch and apply them to another.
32. **How do you sign your Git commits?**
    * To sign Git commits, you can use the **-S** option with the **git commit** command. This option uses GPG (GNU Privacy Guard) to sign the commit, providing a way to verify the authenticity of the commit.
33. **What is the role of Git submodules?**
    * Git submodules allow you to include another Git repository as a subdirectory within your own repository. This is useful for managing dependencies and linking separate projects.
34. **What is the purpose of the git pull --rebase command?**
    * The **git pull --rebase** command fetches changes from a remote repository and rebases your local changes on top of the remote changes. It is an alternative to the regular pull, which merges the changes.
35. **How do you delete a remote branch in Git?**
    * To delete a remote branch in Git, you can use the **git push origin --delete <branch-name>** command. This removes the branch from the remote repository.
36. **What is the Git upstream branch?**
    * The upstream branch in Git refers to the default remote branch that your local branch is associated with. It is the branch you set up to track when you create a new branch or clone a repository.
37. **Explain what git submodule update does.**
    * The **git submodule update** command initializes, fetches, and checks out the appropriate commit in each submodule in a Git repository. It is used to bring the submodules to the state specified by the superproject.
38. **What is the purpose of the git stash command?**
    * The **git stash** command is used to save changes in the working directory that are not ready to be committed. It allows you to switch to another branch or commit without committing or discarding the changes.
39. **How do you rename a local branch in Git?**
    * To rename a local branch in Git, you can use the **git branch -m <new-branch-name>** command. This changes the name of the current branch.
40. **How do you remove untracked files in Git?**
    * To remove untracked files in Git, you can use the **git clean -n** command to preview the files that will be deleted and then use **git clean -f** to actually remove them.
41. **What is the git push --force command used for?**
    * The **git push --force** command is used to forcefully push local changes to a remote repository, overwriting the remote branch's history with the local branch's history. It should be used with caution.
42. **How do you revert a Git repository to a previous commit?**
    * To revert a Git repository to a previous commit, you can use the **git reset** or **git revert** command. The choice depends on whether you want to preserve the commit history or create a new commit that undoes the changes.
43. **What is the purpose of the git describe command?**
    * The **git describe** command provides a human-readable output describing the given commit relative to the closest "annotated" (tagged) commit. It is useful for obtaining a meaningful version string.
44. **Explain the difference between Git and SVN branching models.**
    * **Git:** Uses a distributed branching model, where each developer has their own local branch. Branches are lightweight and easy to create, merge, and delete.
    * **SVN:** Uses a centralized branching model, where branches are created in the central repository. Branching and merging can be more complex and time-consuming.
45. **What is a Git ref and how is it different from a Git branch?**
    * A Git ref (reference) is a pointer to a specific commit, and it can include branch names, tag names, or other references. A branch is a specific type of ref that automatically moves forward as new commits are added.
46. **How do you create an annotated tag in Git?**
    * To create an annotated tag in Git, you can use the **git tag -a <tag-name>** command, which opens the default text editor for you to add a tag message. Annotated tags store extra information such as the tagger's name, email, and date.
47. **What is the purpose of the .gitkeep file?**
    * The **.gitkeep** file is used to include an otherwise empty directory in the version control system. Git ignores empty directories, so adding a **.gitkeep** file helps in keeping the directory tracked.
48. **How do you unstage changes in Git?**
    * To unstage changes in Git, you can use the **git reset HEAD <file>** command. This removes the changes from the staging area while keeping them in the working directory.
49. **What is the git log command used for?**
    * The **git log** command is used to display a log of commits in the repository. It shows information such as commit hashes, author, date, and commit messages.
50. **What is the git reflog command, and how is it used?**
    * The **git reflog** command displays the reference logs, including changes to the branches and HEAD. It helps you recover lost commits, branches, or other references.
51. **How do you amend the last commit in Git?**
    * To amend the last commit in Git, you can use the **git commit --amend** command. This opens the default text editor, allowing you to modify the commit message or add additional changes.
52. **Explain what git fetch does.**
    * The **git fetch** command is used to download changes from a remote repository to your local repository. It updates the remote-tracking branches but does not integrate the changes into your working files.
53. **What is the purpose of the .gitattributes file?**
    * The **.gitattributes** file is used to specify attributes for pathnames in the Git repository. It can be used to control the normalization of line endings, merge strategies, and other attributes.
54. **How can you view the changes between two commits in Git?**
    * To view the changes between two commits in Git, you can use the **git diff <commit1> <commit2>** command. This shows the differences in the content of the specified commits.
55. **What is the role of .gitignore in a Git repository?**
    * The **.gitignore** file specifies intentionally untracked files that Git should ignore. It helps in avoiding the accidental inclusion of unnecessary files in the version control system.