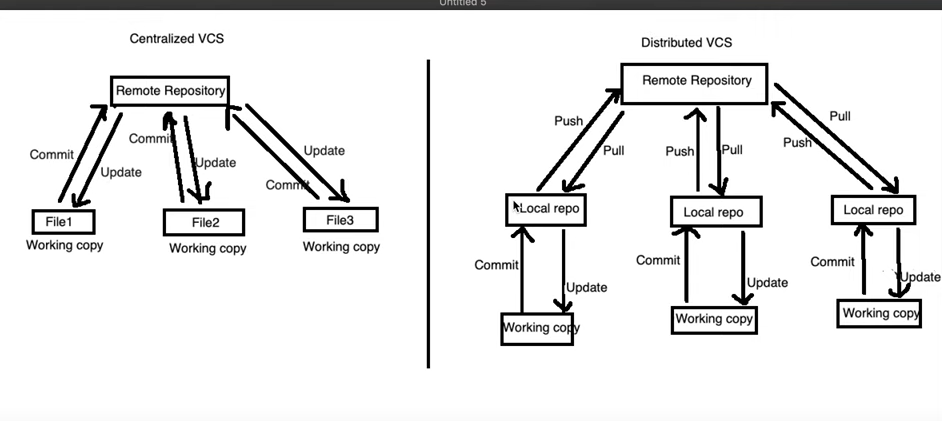
**Questions**

**GIT**

1. [**What is Git?**](https://www.geeksforgeeks.org/what-is-git/)

Git is a [distributed version control system](https://www.geeksforgeeks.org/version-control-systems/) (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?**](https://www.geeksforgeeks.org/what-is-a-git-repository/)

A [Git repository](https://www.geeksforgeeks.org/what-is-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](https://www.geeksforgeeks.org/introduction-to-github/), [GitLab](https://www.geeksforgeeks.org/gitlab/), or [Bitbucket](https://www.geeksforgeeks.org/introduction-to-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

**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 commands 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.

**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 --version -> To check git version installed

Git init . -> To initialize git repository, here . means pwd(present working directory)

Git add \* -> To move all files to staging area

Git add <filename> -> to move one particular file to staging area.

Git commit -m “commit msg” <<filename>> -> to commit or move changes into repository

Git commit =m “commit msg” . -> to move all files to repo from staging area

Git rm –cached <<filename>> -> to move files from staging area back to working directory

Git log -> to see commit full details

Git log –oneline -> to see commit details, it will only display half commit id and msg

Git log –pretty=oneline -> it displays complete commit id and msg

Git log -2 -> displays top 2 commits

Git log -I -> displays top I no.of commits

Git log –follow –all <<filename>> -> to get full commits of a single file

Git config user.name “username” -> to configure username

Git config user. Email “email” -> to configure email id

Git commit –amend -m “message” -> to edit the latest commit message

Git commit –amend –author=”name” -> to edit author for latest commit

Git commit –amend –no-edit -> to commit the changes with previous commit

\*There are 2 ways to delete commits

1. Soft -> git reset –soft HEAD~2 ->deleted 2 commits from head

2. Hard -> git reset –hard HEAD~2 -> delete 1 commit from head along with data

Git update -ref -d HEAD -> to delete all commits

Git reflog -> displays the data of deleted commits

Git cherry-pick <<commit-id>> -> to get back the deleted commit

Git cherry-pick <<commitid1>><<commitid2>> … -> to get back multiple deleted commits

Git branch -> to know in which branch we were in

Git checkout <branchname> -> to switch between branches

Git merge b2 -> if we want to merge code from b2 to b1 then we should be in b1 to merge the code into b1 from b2

Git merge b2 cherry-pick <<commitid1>> <<commitid2>> … -> to merge only specific commit changes to b1 from b2

\*We cannot delete the branch by being in the same branch that we want to delete, to delete a branch first we need to checkout to other branch

Git branch -m <<existing branchname>> <<new branchname>> - command to modify the branch name, -m indicates modify

Git stash -> to delete change temporarily

Git stash apply -> to revert back the changes

Git remote add origin <<github repo url>> -> to link github repo to local system

Git push -u origin <<branch name>> -> to push code into repo

Git remote -v -> to check whether git is connected to any repo or not

Git push -u origin branch1 branch2 branch3 -> to push multiple branches at a time

Git push -u origin –all -> to push all branches at a time

Git pull origin <<branchname>> -> to pull the commits from github

Pull = fetch + merge

Fetch -> checks for the changes from central repo, if changes were present then we will merge to local repo with merge command

Merge -> to get changes from remote repository.

Git fetch origin <<branchname>> -> command to fetch the changes

Git merge origin/<<branchname>> -> command to merge the changes

Git clone url -> to clone the repo

Git remote rm origin -> command to unlink local repo from remote repo

Git revert <<commit id>> -> to delete files that part of this commit id and creates new commit id for deleting the files. The existing commit id won’t be deleted

Git diff <<branchname>> -> used to show difference between branches