**1)What is GitHub?**

**GitHub** is a developer platform that allows developers to create, store, manage and share their code. It uses Git software, providing the distributed version control of access control, bug tracking, software feature requests, task management, continuous integration for every project.

**2)What is Git?**

Git is a distributed version control system, that allows developers to track versions of files and collaborate on software projects. It was originally developed by Linus Torvalds in 2005 and is widely used for both open source and commercial software development.

**3)Stages of Git?**

The three main stages of Git are:

* Untracked: The file exists but is not part of Git's version control
* Staged: The file has been added to Git's version control but changes have not been committed
* Committed: The change has been committed.

**4)Alternates of GitHub?**

There are many alternatives to GitHub, including:

* GitLab

A self-hosted platform with tools for project management and collaboration, plus strong security features and DevOps capabilities

* Bitbucket

A code hosting platform with unlimited free public and private repositories, and strong integration with Atlassian tools

* Azure DevOps

A cloud-based platform from Microsoft with tools for version control, agile planning, CI/CD, and team collaboration

* Source Forge

A community-driven platform with multiple version control options and a focus on open-source projects

* Launchpad

A collaborative development platform for Ubuntu and other open-source projects

* Radicle

A sovereign code infrastructure for secure collaboration over a peer-to-peer network

* Gitea

A lightweight code hosting solution written in Go

* HG Keeper

A Mercurial repository hosting tool that supports HTTP pulling and SSH pushing

* Amazon Code Commit

A fully-managed source control service that offers scalable private Git repositories

* Google Cloud Source Repositories

A cloud code platform that allows users to connect to repositories hosted on GitHub or Bitbucket.

**5)Git config commands?**

Git configuration is a fundamental aspect of setting up and customizing your Git environment. It involves specifying settings that control the behavior of Git across various aspects of the version control process. These configurations can be set at different levels, including system-wide, user-specific, or for a single repository.

**Configuring Git**

The primary tool for configuring Git is the git config command. It allows you to set configuration variables that control aspects like the author name, email address, preferred text editor, diff algorithms, and more. Here's a basic example of using git config:

# Set the name that will be attached to your commits and tags

git config --global user.name "Your Name"

# Set the e-mail address that will be attached to your commits and tags

git config --global user.email "your\_email@example.com"

# Set automatic command line coloring for Git for easy reviewing

git config --global color.ui auto

**Configuration Levels**

Git configurations can be applied at three distinct levels:

* **System level (**--system**)**: Applies settings to every user on the system and all their repositories.
* **User level (**--global**)**: Applies settings to all repositories the current user works with.
* **Repository level (**--local**)**: Applies settings to a specific repository.

Each level overrides settings at the previous level, so repository-specific settings will take precedence over user-level settings, which in turn take precedence over system-wide settings.

**Common Configuration Options**

Some common configuration options include:

* user.name and user.email: Set your name and email address to be used for your commit transactions.
* core.editor: Set your preferred text editor that will be used by commands like git commit.
* merge.tool: Specify the merge tool used by Git when conflicts occur.

**Advanced Configuration**

For more advanced configuration, you can define aliases for common Git commands, set up custom log output formats, or even create hooks for triggering custom scripts during various Git operations. Here's an example of setting an alias:

# Set an alias for a complex log output format

git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short"

**Git Attributes and Hooks**

Git also allows for further customization through **Git attributes** and **Git hooks**:

* **Git Attributes**: Enable you to specify custom attributes (such as merge strategies or diff algorithms) for certain paths in your Git repository.
* **Git Hooks**: Scripts that can run automatically before or after Git commands like commit or push.

**6)Git remote commands?**

**Git remote** is a command used in Git to manage a set of tracked repositories, often referred to as "remotes". A remote in Git is a common repository that all team members use to exchange their changes. In most cases, such a remote repository is stored on a code hosting service like GitHub or on an internal server. Using the git remote command, you can add, view, and manage a repository's remotes.

**Basic Usage of Git Remote**

The git remote command is versatile, offering several subcommands that allow you to add, remove, and manage connections to other repositories. Here are some of the most commonly used subcommands:

* git remote add: This command is used to add a new remote Git repository. For example, git remote add origin https://github.com/user/repo.git adds a remote named "origin" for the specified URL.
* git remote -v: This command lists the remote connections you have to other repositories. The -v or --verbose option shows the URLs of each connection, which can be helpful for checking the paths used for fetching and pushing.
* git remote remove: This command removes a remote from your repository. If you have a remote named "origin" and you want to remove it, you would use git remote remove origin.
* git remote rename: This command allows you to change the name of a remote. For example, if you want to rename a remote from "origin" to "upstream", you would use git remote rename origin upstream.

**Advanced Configuration**

The git remote command also allows for more advanced configurations:

* Setting the default branch for a remote using git remote set-head.
* Changing the list of branches tracked by a remote with git remote set-branches.
* Modifying the URLs associated with a remote using git remote set-url.

**Examples and Practical Uses**

Here are some practical examples of using git remote:

* **Adding a remote**: git remote add origin https://github.com/user/repo.git adds a remote named "origin" pointing to the specified URL.
* **Changing a remote's URL**: git remote set-url origin https://github.com/user/newrepo.git changes the URL for the remote named "origin" to the new specified URL.
* **Pruning dead branches**: git remote prune origin removes any local references to remote branches that have been deleted in the remote repository.

**7)Git add commands?**

The git add command is a fundamental tool in the Git version control system, serving as the first step in the tracking process of changes in the repository. It's used to stage changes for the next commit, meaning that git add tells Git that you want to include updates to a particular file in the next commit. However, it's important to note that git add doesn't affect the repository in any significant way—changes are not actually recorded until you run git commit.

**How to Use git add**

You can stage a specific file, multiple files, or all changes since the last commit. Here are some common uses:

* **Stage a specific file**: git add <file>
* **Stage all changes in the repository**: git add -A
* **Stage changes interactively**: git add -p

For example, if you've edited a file called README.md, you would run the following command to stage it:

git add README.md

If you want to stage every change you've made since the last commit, you would use:

git add -A

And if you want to stage parts of files interactively, you can use:

git add -p

**Best Practices and Considerations**

* **Atomic Commits**: It's a good practice to make atomic commits, which are small and contain a single logical change. git add allows you to stage changes file-by-file or even hunk-by-hunk, helping you create a clean and understandable commit history.
* **Avoiding Unnecessary Files**: Be cautious when using commands that stage many files at once, like git add . or git add -A. You might inadvertently stage files that should be ignored or aren't ready to be committed.
* **Using**.gitignore: To prevent certain files from being staged, you can use a .gitignore file to list patterns of files that Git should ignore.

**Undoing git add**

If you accidentally stage a file, you can unstage it using git reset. For example:

git reset HEAD <file>

This command will unstage the file but leave your changes in the working directory.

**8)Git branch commands?**

The git branch command is a fundamental part of the Git version control system, which is used to manage and store revisions of projects. Branching in Git allows you to diverge from the main line of development and continue to work independently without affecting the main line.

Git Branch Command Usage

Here's how you can use the git branch command:

* To list all branches in your repository, you can simply run: git branch This will display all the local branches in your current repository.
* To create a new branch, you can use: git branch <branch-name> Replace <branch-name> with the desired name for your branch.
* To delete a branch, the command is: git branch -d <branch-name> The -d option deletes the specified branch, but it will refuse to delete if the branch has unmerged changes.
* If you want to force delete a branch, regardless of its merge status, you can use: git branch -D <branch-name> The -D option is a force delete.

**9)Git pull and push commands?**

**Pull command:**

The git pull command is a critical tool for developers, allowing them to update their local repository with changes from a remote repository. It's a combination of two other commands, git fetch followed by git merge, which means it fetches changes from a remote repository and then merges them into the current branch in the local repository.

**Syntax and Usage**

The basic syntax of the git pull command is as follows:

git pull [<options>] [<repository> [<refspec>…​]]

This command fetches changes from the specified remote repository and merges them into the current branch. If no repository is specified, Git will use the default remote, which is typically origin. The <refspec> argument allows you to specify which branches or tags to fetch and merge.

**Push command:**

**Git push** is a command in Git that allows you to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo. It's the counterpart to **git fetch**, but whereas fetching imports commits to local branches, pushing exports commits to remote branches.

**How Git Push Works**

When you create commits on your local branch, you're adding new history to the repository. However, this new history is only available to you until you share it with the same branch on the remote repository. The git push command takes the commits from your local branch and adds them to the corresponding branch on the remote repository.

Here's a basic example of using git push:

git push <remote> <branch>

* <remote> is the alias of the remote repository. Typically, this is origin, which is the default name Git gives to the server you cloned from.
* <branch> is the name of the remote branch where you want to push your changes.

If you're pushing to the same branch name on the remote and your current branch is set to track the remote branch, you can simply use:

git push.

**10)Git init commands?**

The git init command is a fundamental Git operation used to start a new repository. It's the first command you run to initialize a new Git repository or to reinitialize an existing one.

**Command Syntax and Usage**

The basic syntax for git init is as follows:

git init [options] [directory]

When you run git init, Git creates a hidden .git directory in the specified directory. This .git directory contains all the necessary metadata for the repository, including objects, refs, and configuration files. If no directory is specified, git init initializes the current directory as a Git repository.

**Options and Configuration**

git init comes with several options that allow you to customize the initialization process:

* -q or --quiet: Suppresses output except for errors and warnings.
* --bare: Initializes the repository as a bare repository, which is suitable for server-side repositories that do not require a working directory.
* --template=<template-directory>: Specifies a directory from which to take repository templates.
* --separate-git-dir=<git-dir>: Creates a text file containing the path to the actual repository, acting as a symbolic link to the .git directory.
* -b <branch-name> or --initial-branch=<branch-name>: Specifies the name of the initial branch in the new repository.
* --shared[=<permissions>]: Sets up the repository to be shared among multiple users with specific permissions.

**11)Upstream commands?**

An **upstream branch** in Git refers to a branch that serves as a reference point for another branch. Typically, it’s used to track the remote branch, allowing developers to fetch updates, compare changes, and push their commits easily. Setting an **upstream branch**simplifies the process of keeping local and remote repositories in sync.

**12)Git shortcuts?**

Git Bash is a command-line application for Windows that emulates the Git command line experience. It's particularly useful for users who are familiar with Unix-based systems but are working on Windows, which traditionally uses a non-Unix terminal environment. Git Bash provides the Unix style command-line functionality which is absent in Windows by default.

**Basic Git Bash Commands**

Here are some basic commands that you can use in Git Bash:

* **pwd**: This command prints the present working directory, which is the folder path where the current Bash session is located.
* **ls**: This command lists the contents of the current working directory, similar to the DIR command in Windows Command Prompt.
* **cd**: Short for 'Change Directory', this command is used to change the current working directory to another directory.

**Core Git Commands in Git Bash**

Git Bash also includes a full set of core Git commands. These are the commands you would use for various Git operations, such as:

* **git clone**: This command is used to clone a repository into a new directory.
* **git commit**: This command is used to record changes to the repository.
* **git checkout**: This command is used to switch branches or restore working tree files.
* **git push**: This command is used to update remote refs along with associated objects.

**Additional Commands**

Git Bash comes with additional Unix commands that are not directly related to Git operations but can be useful in the Git Bash environment. Some of these commands include ssh, scp, cat, and find.