**Git-GitHub**

**Git** is a distributed version control system used to track changes in source code during software development.

It helps developers or testers collaborate and maintain a history of code changes.

Git can be used to manage projects of any size and enables features like branching, merging, and version history.

*Key Features of Git:*

1. **Version Control**: Tracks changes to files and stores a history of revisions.
2. **Distributed System**: Every developer has a full copy of the project repository, including its history.
3. **Branching and Merging**: Allows the creation of branches for different features or bug fixes and later merging them back into the main branch.
4. **Staging Area**: Changes can be reviewed and staged before committing to the repository.
5. **Lightweight and Fast**: Efficient performance even for large projects.

*Common Git Commands:*

* git init: Initialize a new Git repository.
* git clone: Clone an existing repository.
* git add: Stage changes for a commit.
* git commit: Commit changes with a message.
* git push: Push local commits to a remote repository.
* git pull: Fetch and merge changes from a remote repository.
* git branch: Manage branches.
* git merge: Merge branches.

**GitHub:**

**GitHub** is a cloud-based platform that hosts Git repositories.

It provides tools for version control, collaboration, and project management.

Developers or Testers use GitHub to store their code online, collaborate with teams, and manage their development lifecycle.

*Key Features of GitHub:*

1. **Repository Hosting**: Store Git repositories in the cloud.
2. **Collaboration**: Multiple developers can work on the same project using pull requests and code reviews.
3. **Issue Tracking**: Manage bugs, feature requests, and tasks.
4. **Actions (CI/CD)**: Automate workflows like testing, building, and deploying code.
5. **Documentation**: Add README files, wikis, and project details.
6. **Community**: Share code with the open-source community and collaborate globally.

*Common Terminology in GitHub:*

* **Repository (Repo)**: A storage space for your project files and Git history.
* **Pull Request**: A request to merge code from one branch to another, often followed by a code review.
* **Fork**: A copy of someone else's repository that you can modify.
* **Star**: Bookmark a repository to show interest or keep track of it.
* **Issues**: A way to track tasks, enhancements, or bugs.
* **GitHub Actions**: Automate workflows for testing and deploying code.

**Git vs GitHub:**

|  |  |
| --- | --- |
| **Git** | **GitHub** |
| 1) Git is a **version control system.** | 1) GitHub is **a web-based application hosting the servers for Git repositories.** |
| 2) Git is a **tool** that allows you to manage & track the changes in project source code. | 2) GitHub is a **platform** that allow you store or upload all our git repositories to share & collaborate with others. |
| 3) Git **needs to be installed** on a local machine. | 3) GitHub **does not need to be installed** as it is a web-based application. |
| 4) It is accessible through **installation and commands**.  5) Git is used for **tracking** the code changes.  It helps the developer and tester for **branching**, **merging** & **collaborating the history of code changes**, | 4) It is accessible through a **web browser** or Git commands.  5) GitHub is used for **collaborating**, **hosting** & **managing** Git repositories.  It helps developers & testers to store their code online, collaborate with teams, and manage their development cycle. |

| **Feature:** | **Git** | **GitHub** |
| --- | --- | --- |
| **Type:** | Tool (Version Control System) | Platform (Git Repository Hosting) |
| **Location:** | Local (on your computer) | Cloud-based |
| **Usage:** | Version control, branching, and merging | Collaboration, hosting, and community |
| **Commands:** | Requires Git installed locally | Accessible through a browser or API |

**=====================================================================================**

## **GIT Commands**

**1. git init: Initialize a new Git repository.**   Example: `**git init**`

**2. git clone: Clone a remote repository to your local machine.**   Example: `**git clone https://github.com/user/repo.git**`

**3. git add: Add files or changes to the staging area.**   Example:  `**git add \***` - To add all the changes to staging area

`**git add <file-path>**` - To add single file to staging area

`**git add <file1-path> <file2-path>**` - To add multiple files to staging area

**4. git commit: Commit the staged changes to the repository.**   Example: `**git commit -m "Commit message**` 👉 Commits **staged changes** to the local repository.

  `**git commit -a -m "Commit message**` 👉 **Automatically stages** all **modified tracked files** from the working directory and then commits them to the local repository.

**5. git status: Show the current status of the repository.**   Example: `**git status**`

**6. git log: View the commit history.**   Example: `**git log**`

**7. git diff: Show the differences between files or commits.**   Example: `git diff file.txt`

**8. git reset: To reset/removed uncommitted changes.**

Example:  `**git reset --hard**` - To undo/remove all the uncommitted changes in all files permanently

`**git checkout HEAD -- <file-path>**`- To undo/remove uncommitted changes in a specific file.

**9. git reset: To reset/removed committed changes.**

Example: `**git reset --soft HEAD~1**` - To reset/undo the commit but keep changes in staging area

`**git reset --hard HEAD~1**` - To reset/undo the commit and remove changes from staging area

**-----------------------------------------------------------------------------------------------------------------------------------------------------**

git branch: **# List all existing local branches & show current branches.**

git branch <new-branch-name>: **# Create a new branch.**

git checkout <branch-name>: **# Switch to an existing branch**

git switch <branch-name>: **# (modern) Switch to an existing branch**

git checkout -b < new-branch-name >: **# Create and switch to a new branch**

git switch -c < new-branch-name >: **# (modern) Create and switch to a new branch**

git push origin <branch-name>: **# Push a single local branch to the remote repository**

**git push --all origin: # Push all local branches to the remote repository**

git branch -d <branch-name>: **# Delete a local branch (only if merged)**

git branch -D <branch-name>: **# Force delete a local branch (even if unmerged)**

git push origin --delete <branch-name>: **# Delete a branch from the remote (e.g., GitHub)**

git branch -m <new-branch-name>: Rename the current branch

git branch –v : See last commit on each branch

git branch –merged : See which branches are merged into current branch

**8. git branch: List, create, or delete branches.**   Example: `git branch branchname`

**9. git checkout: Switch to a different branch.**   Example: `git checkout branchname`

**10. git merge: Merge changes from one branch into another.**    Example: `git merge branchname`

**11. git remote: Manage remote repositories.**    Example: `git remote add origin https://github.com/user/repo.git`

**12. git push: Push local changes to a remote repository.**    Example: `git push origin branchname`

**13. git pull: Fetch and merge changes from a remote repository.**    Example: `git pull origin branchname`

**14. git fetch: Fetch changes from a remote repository.**    Example: `git fetch origin`

**15. git stash: Save changes that are not ready to be committed.**    Example: `git stash`

**16. git stash pop: Apply the most recent stash and remove it from the stash list.**    Example: `git stash pop`

**17. git reset: Reset the repository to a previous commit.**    Example: `git reset commit\_hash`

**18. git revert: Create a new commit that undoes a previous commit.**    Example: `git revert commit\_hash`

**19. git tag: Create and manage tags for marking specific points in history.**    Example: `git tag v1.0.0`

**20. git remote -v: View the URLs of the remote repositories.**    Example: `**git remote -v**`

**21. git config: Set or get repository options.**    Example: `git config --global user.name "Your Name"`

**22. git show: Show the details of a specific commit.**    Example: `git show commit\_hash`

**23. git cherry-pick: Apply a specific commit from one branch to another.**    Example: `git cherry-pick commit\_hash`

**24. git rebase: Reapply commits on top of another base commit.**    Example: `git rebase branchname`

**25. git blame: Show who changed which lines in a file.**    Example: `git blame file.txt`

**26. git remote add: Add a new remote repository.**    Example: `git remote add origin https://github.com/user/repo.git`

**27. git remote remove: Remove a remote repository.**    Example: `git remote remove origin`

**28. git log --oneline: Show the commit history in a condensed format.**    Example: `git log --oneline`

**29. git log --graph: Show the commit history in a graphical representation.**    Example: `git log --graph`

**30. git log --author: Show the commit history by a specific author.**    Example: `git log --author "John Doe"`

**31. git blame: Show who changed which lines in a file.**    Example: `git blame file.txt`

**32. git branch -d: Delete a branch.**    Example:

 `git branch -d branchname`

**33. git branch -m: Rename a branch.**    Example: `git branch -m new\_branchname`

**34. git show-branch: Show branches and their commits.**    Example: `git show-branch`

**35. git clean: Remove untracked files from the working directory.**    Example: `git clean -f`

**36. git remote prune: Remove remote-tracking branches that no longer exist on the remote.**    Example: `git remote prune origin`

**37. git log --grep: Show the commit history that matches a specific pattern.**    Example: `git log --grep "bug fix"`

**38. git log --since: Show the commit history since a specific date.**    Example: `git log --since "2022-01-01"`

**39. git log --until: Show the commit history until a specific date.**    Example: `git log --until "2022-12-31"`

**40. git bisect: Find the commit that introduced a bug using binary search.**    Example: `git bisect start`

**41. git reflog: Show a log of all reference changes in the repository.**    Example: `git reflog`

**42. git remote show: Show information about a remote repository.**    Example: `git remote show origin`

**43. git revert --no-commit: Revert changes but do not create a new commit.**    Example: `git revert --no-commit commit\_hash`

**44. git reset --hard: Discard all changes and reset the repository to a specific commit.**    Example: `git reset --hard commit\_hash`

**45. git config --global alias: Set up an alias for a Git command.**    Example: `git config --global alias.ci commit`

**46. git archive: Create a tar or zip archive of a Git repository.**    Example: `git archive --format=zip --output=archive.zip master`

**47. git submodule: Manage Git submodules within a repository.**    Example: `git submodule add https://github.com/user/repo.git`

**48. git clean -n: Dry run of git clean to preview files that will be removed.**    Example: `git clean -n`

**49. git log --follow: Show the commit history of a renamed file.**    Example: `git log --follow file.txt`

**50. git show-branch --all: Show the commit history of all branches.**    Example: `git show-branch --all`