**Git Important Commands & Descriptions**

1. What is Git?

Git is a Version Control System and a tool that helps to track changes in code. It is popular, free & open source, fast & scalable.

1. Features of Git.
   1. Track the changes.
   2. Collaborate
2. What is GitHub?

A website that allows developers to store and manage their code using Git.

1. Git Branching Strategies follows in the Tech Industry.

There are 4 different type of branches –

* 1. Master/Main branches – This is the default branch available in the repository.
  2. Feature branches – We can take a branch from Main/Master branch and do the development for the feature.
  3. Release branches – After every feature added to the main branches in every 1-3 months a company had introduced some Releases which can be called as versions. So, from the main branch a new branch gets created that is Release branch.
  4. Hotfix/Patchfix branches – This happens mostly after the releases when some user might have reported any issue/bug. Developer has to fix that bug, so far that we create Hotfix/Patchfix branches.

1. A list of all commands along with short descriptions is provided below:

|  |  |
| --- | --- |
| **Commands** | **Short description** |
| git init | To initialize a local repo as a git repo. |
| git config --global user.name “YOUR\_NAME” | To config your name globally. |
| git config --global user.email “YOUR\_EMAIL” | To config your email globally. |
| git config --list | List out the git configuration. |
| git clone <GITHUB\_REPO\_URL> | Clone the GitHub repository into your local. |
| git clone <GITHUB\_REPO\_URL> . | Added a **.** so that it won’t create another directory inside your local repo. Used just to copy the repository. |
| git status | Check the status of the code.  (**Untracked(New), Modified(Changed), Staged(Ready to be commit), Unmodified(Unchanged)**) |
| git add <FILENAME> | Stages the specific filename. |
| git add . | Stages all new and modified files, without delete. |
| git add -A | Stages all. |
| git add -u | Stages all modified and deleted files, without new. |
| git commit -m “YOUR\_COMMIT\_MESSAGE” | Commit the changes, after the staging done. |
| git push origin main | Push the content to your Github repository into main branch. |
| git remote add origin <GitHub URL Link> | Add the origin, so that when you push code, it will push the code correctly to your GitHub repository. |
| git remote -v | Verify the remote origin. |
| git remote set-url origin <GitHub URL Link> | Modify the origin URL. |
| git push -u origin main | Push command, with the upstream. Means we are telling git that afterwards, I will push all the changes to the origin main branch only. Next time, **git push** commandis enough. |
| git branch | Verify the branch/List of branches available. |
| git branch -M main | Rename the branch. By default, **master** is the branch in working directory, you can rename it to **main**. |
| git checkout <BRANCH\_NAME> | Navigate branch. |
| git checkout -b <NEW\_BRANCH\_NAME> | Create new branch and switch to it immediately. |
| git branch -d <BRANCH\_NAME> | Deletes a local branch that has already been merged into the current branch. |
| git branch -D <BRANCH\_NAME> | Forces the deletion of a local branch, even if it hasn’t been merged. |
| git push origin --delete <BRANCH\_NAME> | Deletes a remote branch. |
| git push –force origin <BRANCH\_NAME> | Forces the push of your changes, overwriting the remote branch. **Use with caution because it can rewrite history.** |
| git pull origin main | Fetches and merges changes from the remote repository to your local branch. |
| git log | Display the commit history of the repository, showing commits along with messages, author information, and dates. |
| git log --online | Displays the commit history in a condensed, one-line format. |
| git show <COMMIT\_HASH> | Displays detailed information about a specific commit, including the difference for that commit. |
| git diff | Shows unstaged changes in the working directory. |
| git diff --cached/--staged | Shows staged changes ready to be committed. |
| git diff <OLD\_COMMIT> <NEW\_COMMIT> | Shows differences between two commits |
| git diff branch1..branch2 | Shows differences between two branches. |
| git diff <FILENAME> | Shows changes in a specific file. |
| git diff --word-diff | Shows difference word-by-word within lines. |
| git diff <COMMIT-HASH> | Shows difference from your working directory and the mentioned commit. |
| git merge <BRANCH\_NAME> | Merges the specified branch into the current branch. **Note*:* You need to switch to the branch you want to merge into before running this command.** |
| git rebase <BRANCH\_NAME> | Switch to the branch you want to rebase(feature-branch): **git checkout feature branch**  Rebase your branch onto the target branch (eg. main)  **git rebase main** |
| git reset <FILENAME> | Unstage a file that has been added to the staging area but does not modify the file itself. |
| git checkout <FILENAME> | Reverts changes in a specific file to the version in the last commit (undoes local changes to a file). |
| git checkout . | Modify all the files to the last commit. |
| git reset | To unstage all changes (Remove all files from the staging area). |
| git reset --soft HEAD~1 | Undo the last commit but keep changes staged. |
| git reset --mixed HEAD~1 | Undo the last commit and unstage changes, but keep them in the working directory. |
| git reset --hard HEAD~1 | Undo the last commit and discard changes completely (both staged and working directory). |
| git reset --hard <COMMIT\_HASH> | Reset to a specific commit, discarding all changes after that commit (**be cautious with this**) |
| git fetch | Download new commits from the remote repository but does not automatically merge them. Use this before checking out a branch or merging. |
| git rm -r –cached <FILENAME> | Remove the **specified** file from the cached/tracked data. |
| git stash | Stashed your changes (both staged and unstaged) |
| git stash -k/--keep-index | Stashed only unstaged changes, keeps staged changes |
| git stash save “MESSAGE” | Stashed with a custom message. |
| git stash list | List all stashed changes. |
| git stash apply | Applies the latest stash without removing it from the stash list. |
| git stash apply stash@{1} | Applied the specific stash to bring it back into the working directory. |
| git stash pop | Applies the latest stash and removes it from the stash list. |
| git stash drop stash@{0} | Removes a specific stash from the list |
| git stash clear | Clears all stashed changes. |
| git stash -u/--include-untracked | Stashes untracked files as well. |
| git stash -a/-all | Stashes untracked and ignored files. |
| git clean -f | Removes untracked files in your working directory. |
| git tag <TAG\_NAME> | Creates a tag for a specific commit (usually to mark releases). |
| git push origin <TAG\_NAME> | Pushes a specific tag to the remote repository. |
| git tag -d <TAG\_NAME> | Deletes a local tag. |
| git help <COMMAND> | Displays help information for a specific Git command. |

1. Difference between **git merge** and **git rebase**?

|  |  |  |
| --- | --- | --- |
| **Feature** | **git merge** | **git rebase** |
| **History** | Creates a merge commit, preserving all commit history. | Rewrites commit history, creating a linear history |
| **Merge Commits** | Create a merge commit | No merge commits, just a linear history. |
| **Conflict resolution** | Resolve conflicts once in the merge commit | Resolve conflicts as you replay each commit. |
| **Use Case** | Suitable for preserving commit history of both branches | Suitable for creating a clean, linear history. |

1. Difference between Git and GitHub?

|  |  |  |
| --- | --- | --- |
| **Feature** | **Git** | **GitHub** |
| **Type** | Version control system (VCS) | Cloud-based hosting platform for Git repositories. |
| **Use Case** | Track and manage versions of code locally. | Collaborate and host Git repositories online. |
| **Primary Function** | Manages and tracks changes in source code. | Provides remote storage for Git repositories, collaboration tools, and social feature. |
| **Usage** | Command-line tool for local repository management. | Web-based platform with additional collaboration tools (PRs, issues etc.) |
| **Offline Capability** | Fully functional offline (except for pushing/pulling to/from remotes) | Requires an internet connection to interact with repositories. |
| **Branching/Merging** | Provides tools for local branching and merging | Allows for pull requests are code review features for merging into main projects. |
| **Hosting** | Does not host repositories itself. | Hosts repositories and adds collaboration tools like PRs, issues, and documentation. |

1. How to create a Pull Request on GitHub?

Steps to raise Pull Request (PR) on GitHub:-

* 1. Fork the Repository (if applicable).
     1. If you’re not a direct collaborator, you’ll need to fork the repository. This creates a copy of the repository under your GitHub account.
     2. After forking, clone your repository locally:

**git clone** [**https://github.com/YOUR-USERNAME/REPO.NAME.git**](https://github.com/YOUR-USERNAME/REPO.NAME.git)

* 1. Create a new branch for your changes.
     1. It’s a best practice to create a new branch for each feature or fix:

**git checkout -b <FEATURE-BRANCH>**

* + 1. This keeps your changes separate from the main branch and organized.
  1. Make and commit your changes.
     1. Edit your files, then stage and commit them:

**git add .**

**git commit -m “DESCRIPTION OF THE CHANGES MADE”**

* 1. Push your branch to your remote repository.
     1. Push the feature branch to your GitHub repository:

**git push origin FEATURE-BRANCH**

* 1. Open a Pull request on GitHub.
     1. Go to your GitHub repository in a web browser.
     2. You’ll see a prompt to create a pull request for your recently pushed branch, or you can navigate to the Pull Requests tab and click New Pull Request.
     3. Choose your branch (FEATURE-BRANCH) as the source and the branch you’re merging into (e.g., main) as the destination.
  2. Fill out the Pull request details.
     1. Give your pull request a title that briefly describe the changes.
     2. Add a description of the changes you made, why they’re needed, and any additional context or instructions for reviewers.
     3. Mention specific people for review, if needed, by tagging them (e.g., @USERNAME).
  3. Submit the Pull request.
     1. Click **Create Pull Request** to submit the PR.
     2. Your changes will now be visible, and reviewers can leave comments, approve, or request changes before merging.