Git & GitHub

What is Git & GitHub ?

- Git is a **Version Control System (VCS)** that:
  - Keeps careful track of changes in our files, documenting who made changes and when they were made.
  - Facilitates collaboration with others on projects by managing and merging contributions from multiple people.
  - Allows testing of changes without losing the original versions, ensuring that we can experiment safely.
  - Provides the ability to revert back to older versions when needed, enabling recovery from errors or unwanted changes.

- GitHub is a **Web-based hosting service** for git, that:
  - Provides a "remote" location for storing the git workspaces.
  - Useful if local system gets in any accidental events, the works will be still available.

Types of Repository

Local Repository

A **local repository** is a version-controlled directory on your local machine. It contains all the files, commits, branches, and history for your project. You interact with a local repository using Git commands, such as adding files, committing changes, creating branches, and more.

Remote Repository

A **remote repository** is a version-controlled repository hosted on a server, often accessed via the internet. It serves as a centralized location where collaborators can push their local changes and pull updates from others. Examples of services hosting remote repositories include GitHub, GitLab, Bitbucket, and others.

GitHub Repository

A **GitHub repository** is a specific type of remote repository hosted on GitHub. It provides a web-based interface for managing your repository and offers additional features like issues, pull requests, project boards, and more.

Git Commands

- **git init**
  This command initialize an existing directory as a Git repository.

- **git clone <url>**
  Downloads an existing repository from GitHub and creates a synced, local copy. Here **"<url>"** specifies the URL of the remote repository to be cloned.

- **git add <filename>**
  Signals to git that the specified file(**<filename>**) should be tracked for changes and places modified files in the staging area. Files not added in this way are ignored by git.
  `**Git add .**` Commands Git to take a snapshot of the contents of all files under the current directory.

- **git reset <filename>**
  This command is used to unstage a file(**<filename>**) that has been added to the staging area, but without modifying the working directory.

- **git status**
  Displays useful information about repository (e.g., current branch, tracked/untracked files, differences between local and remote versions) and modified files that are staged for next commit.

- **git diff**
  shows the differences between the files in your working directory and the staging area. It highlights what changes have been made to files that are not yet staged for the next commit.
  `**git diff --staged**` This command shows the differences between the files in the staging area and the last commit. It highlights what changes are staged and will be included in the next commit.

- **git commit -m "message"**
  Takes a "snapshot" of all files currently on the staging area and commits it to git's memory. The "snapshot" is captioned with the given **"message"** as a brief description for the commit.

- **git log**
  This command shows commit history of current active branch. It gives information about, commit hashes(unique identifiers for each commit), author names and email addresses, dates and times of the commits, commit messages.
  ` **git log -p**` This tells Git to include the diff (patch) for each commit in the log output.
  ` **git log --stat --summary**` Here the –stat option displays a summary of changes for each commit.
  The --summary option provides some additional information.
  `**git log --follow <filename>**` command is used to show the commit history of a specific file, including its history across renames and moves. This command helps you trace the changes to a file even if its name or location has changed over time.

- **git rm <filename>**
  Used to remove a file(**<filename>**) from both the working directory and the staging area. This will unstage and remove the file from the repository while leaving it in your working directory.
  After this, we need to commit the changes to finalize the removal: `**git commit -m "Remove <filename>"**`.

- **git mv <source> <destination>**
  This command is used for moving a file from one directory to another. This is also used for renaming a filename by running this command: `**git mv <filename> <rename filename>**`.

- **git fetch <remote>**
  This command downloads commits, files, and references from a remote repository(**<remote>**) into local repository. It updates remote-tracking branches but does not modify working directory or local branches.
  For example, `**git fetch origin**` fetch updates from the default remote repository named origin. Here 'origin' is the default name given to the remote repository from which a local repository was cloned.

- **git merge <remote>/<local-branch>**
  This command merges a remote branch(**<remote>**) into a local branch, so that it is upto date. For example, `**git merge origin/main**`.

- **git pull <remote> <branch>**
  used to fetch and integrate changes from a remote repository into your current branch. It is essentially a combination of two commands: `**git fetch**` followed by `**git merge**`.

- **git push <remote> <branch>**
  Uploads local commits in local repository to the remote repository (i.e., from our computer to GitHub).

- **git branch**
  Used to list, create, rename, and delete branches within a repository.
  `**git branch**` lists all branches in our repository.
  `**git branch <branch-name>**` Used to create a new branch based on the commit we are currently on.
  `**git branch -m <current-branch-name> <new-branch-name>**` Used for renaming a branch.
  `**git branch -d <branch-name>**` Deletes a branch. Git will prevent from deleting the branch if it has unmerged changes. For force delete -D is used.
  `**git branch -r**` Lists remote branches.

- **git checkout <branch>**
  This command is used in Git to switch between different branches or to restore files from a commit or a branch.
  `**git checkout -b <new-branch-name>**` is used to create and switch to the new branch in one step.

Git Branching and Merge conflict

Git branching is a powerful feature that allows developers to work on different aspects of a project simultaneously without interfering with each other's work. However, when branches are merged back into the main branch or into each other, conflicts can arise if Git cannot automatically reconcile the changes. Therefore handling merge conflict is a crucial part.

Branching is done by the command `**git branch**`(mentioned above in 'Git Commands' section).

Merge conflict typically happens when:

- Changes are made to the same lines of a file in both branches.
- One branch deletes a file or changes its structure, while the other branch modifies the same file.

Basically two or more write operations are done on the same file simultaneously. Thus violating the concurrency protocol.

When a merge conflict occurs, Git will pause the merge process and mark the conflicted files. The developer has to resolve the conflicted files, then stage them for commit.