Introduction to Git

Task Assigned by Kharagpur Open Source Society

What is Git?

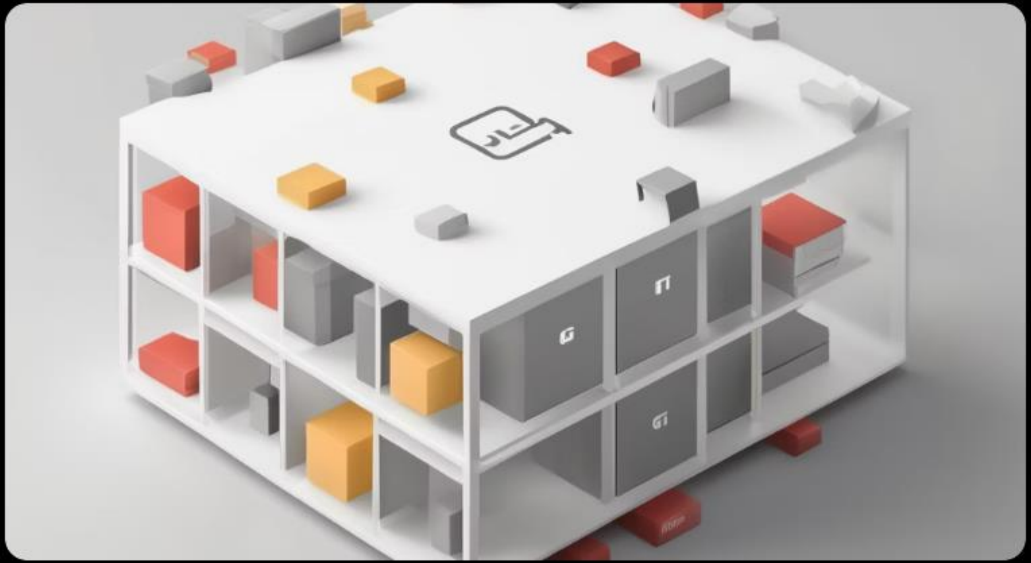
Version Control System

Git is a distributed version control system that allows multiple developers to work on a project simultaneously. It keeps track of changes made to files and allows developers to collaborate efficiently.

Importance of Git

- Git enables developers to track changes, revert to previous versions, and collaborate effectively.
- It provides a centralized repository where developers can store and manage their code.
- Git also allows for branching and merging, making it easier to work on different features or versions of a project simultaneously.

Elaboration of Commands



git init

- Initializes a new Git repository in the current directory.
- Use this command when starting a new project to create a new repository.

git add

- Adds changes in the current working directory to the staging area.
- Use this command to prepare changes for a commit.

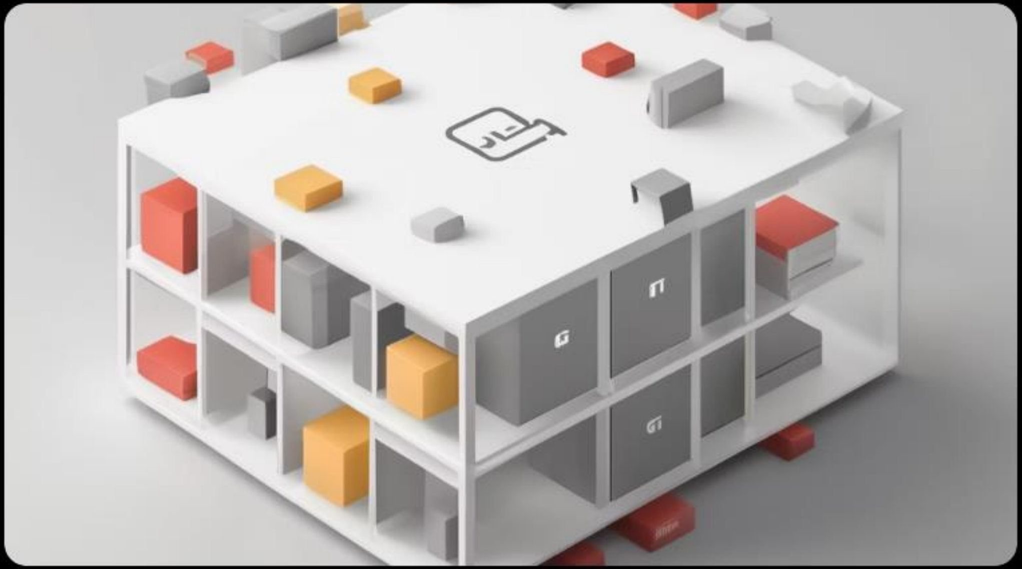
git commit

- Records changes to the repository.
- Use this command to save changes with a descriptive message.
- `git commit -m "Add new feature"`

git push

- Sends local commits to a remote repository.
- Use this command to publish your changes to a shared repository.

Elaboration of Commands



git pull

- Fetches changes from a remote repository and merges them into the current branch.
- Use this command to update your local repository with the latest changes.

git branch

- Lists existing branches or creates a new branch.
- Use this command to manage branches in your repository.

git checkout

- Switches to a different branch or restores files from a specific commit.
- Use this command to navigate between branches or revert changes.

git merge

- Combines changes from different branches into the current branch.
- Use this command to integrate changes from one branch into another.

Git Stash

- Git stash allows you to temporarily store changes that are not ready to be committed.
- Useful for switching tasks or branches without committing unfinished work.
- Git stash temporarily saves your changes and reverts your working directory to the last commit, allowing you to switch branches or work on other tasks without any conflicts.
- Example:

```
git stash
```

```
git checkout <branch>
```

```
git stash pop
```

Git Bisect

- Git Bisect is a command in Git that helps you find the specific commit that introduced a bug.
- It uses a binary search algorithm to efficiently narrow down the range of commits to search through.
- By marking commits as either good or bad, Git Bisect can automatically traverse through the commit history and pinpoint the exact commit where the bug was introduced.
- Example:

```
git bisect start
```

```
git bisect bad
```

```
git bisect good <commit>
```

Git Reflog

- Git reflog records the history of HEAD movements, helping you recover lost commits or undo changes.
- It provides a detailed log of all the recent operations performed on the repository, including commits, branch creations, merges, and more.
- The reflog is useful for recovering lost commits, branches, or other changes that may have been accidentally deleted or modified.
- Example:

  `git reflog`

  `git checkout HEAD {1}`

Git Diff

- Git Diff is a command in Git that is used to view the differences between commits, branches, and files.
- It provides a detailed overview of the changes made, allowing users to understand the modifications and track the history of their code.
- Example:

```
git diff <branch>
```

```
git diff <commit1> <commit2>
```

Git Switch

- Git Switch is a command in Git that allows you to switch between branches in a Git repository.
- It is used when you want to work on a different branch or access the contents of a specific branch.
- The Git Switch command is a convenient way to navigate between branches without creating a new branch.
- Example:

```
git switch <branch>
```

Git Rebase

- Git rebase is a command used to integrate changes from one branch into another.
- It allows you to move or combine a sequence of commits to a new base commit, resulting in a linear commit history.
- This can be useful for keeping your branch up to date with the latest changes from another branch, or for cleaning up your commit history before merging your changes into the main branch.
- Example

  ```
  git rebase <branch>
  ```

- This will take the commits from the current branch and apply them on top of the specified branch. Git will automatically detect any conflicts and prompt you to resolve them before continuing with the rebase.
- It's important to note that rebasing rewrites the commit history, so it should only be used on branches that haven't been pushed to a remote repository or shared with other developers. If you're working on a branch that is being used by others, it's generally better to use the git merge command instead.

Git Cherry-pick

- Git cherry-pick is a command used to apply a specific commit from one branch to another.
- It allows us to select a single commit and apply it to a different branch, without merging the entire branch.
- This can be useful when you want to bring in a specific change or fix from one branch to another, without affecting other changes in the branch.
- Example:

```
git cherry-pick <commit>
```

Live Demo

Git Initialization

- Start by creating a new Git repository using the `git init` command.
- This initializes an empty Git repository in the current directory.

Adding Files

- Use the `git add` command to add files to the staging area.
- For example, `git add file.txt` adds a file named `file.txt` to the staging area.

Committing Changes

- After adding files, use the `git commit` command to create a new commit.
- Include a descriptive message with the commit using the `-m` flag.

Branching

- Create a new branch using the `git branch` command.
- For example, `git branch feature` creates a new branch named `feature`.

Switching Branches

- Switch between branches using the `git checkout` command.
- For example, `git checkout feature` switches to the `feature` branch.

Live Demo

Merging Branches

- Merge branches together using the `git merge` command.
- For example, `git merge feature` merges the `feature` branch into the current branch.

Remote Repositories

- Connect to a remote repository using the `git remote` command.
- For example, `git remote add origin <url>` adds a remote repository named `origin`.

Pushing Changes

- Push local commits to a remote repository using the `git push` command.
- For example, `git push origin main` pushes commits to the `main` branch of the remote repository.

Pulling Changes

- Pull changes from a remote repository using the `git pull` command.
- For example, `git pull origin main` pulls changes from the `main` branch of the remote repository.

Cloning a repo

- Cloning creates a local copy of a remote repository.
- It allows for working on the repository locally without directly modifying the remote repository.

```
git clone https://github.com/username/repository.git
```

Conclusion

- Understanding these Git concepts and commands will empower us to effectively manage version control in our projects.
- Experimentation and practice are crucial for mastering Git.
- Thanks KOSS for helping me to learn such a important tool for contributing in open source.

Thank you