1. BASIC TERMS OF GIT

1. Version Control System (VCS):

Git is a VCS (distributed) that helps to manage and track changes to code and other files over time.

It allows multiple developers to work on a project simultaneously.

2. Repository (Repo):

Repository is a storage space where your project is stored, including its files and the history.

3. Commit:

Commit is a snapshot of the changes made to files in the repository at a specific time.

It is accompanied by a commit message that describes the changes.

4. Branch:

A branch is an independent line of development within a repository.

It allows developers to work on different features and fixes without interfering with each other.

5. Clone:

Cloning is the process of creating a copy of a remote repository on your local machine.

6. Pull:

The process of fetching changes from a remote repository and merging them into your local branch.

7. Push:

- Pushing is the process of sending your committed changes to a remote repository.

8. Merge:

Itcombines changes from different branches, creating a unified history.

9. Conflict:

A conflict occurs when Git is unable to automatically merge changes from different branches, and manual intervention is required.

10. Fetch:

Fetching is the process of downloading changes from a remote repository without merging them into your local branch.

11. Remote:

A remote is a repository hosted on a server, typically on platforms like GitHub, GitLab, or Bitbucket.

12. Pull Request (PR):

In Git-based platforms like GitHub, a pull request is a way to propose changes to a repository. It allows others to review and discuss the changes before merging.

13. Fork:

-Forking is a process where you create a personal copy of someone else's repository. It allows you to freely experiment with changes without affecting the original project.

14. Merge Conflict:

A merge conflict occurs when Git cannot automatically merge changes and requires manual resolution.

2. BASIC COMMANDS OF GIT

1.Initializing a Repository:

git init

2.Cloning a Repository:`

git clone <repository-url>

3.Checking Status:

git status

4.Adding Changes to Staging Area:

git add <filename>

git add or git add –all

5. Committing Changes:

git commit -m "Commit message"

6.Viewing Commit History:

`git log`

7. Creating a Branch:

- `git branch <branch-name>`: Creates a new branch.

git checkout -b <branch-name>`: Creates and switches to a new branch in one command.

8.Switching Between Branches:

- `git checkout <branch-name>`

- `git switch <branch-name>`

9. Merging Changes:

- `git merge <branch-name>`

10. Pulling Changes from a Remote Repository:

- `git pull`

11. Pushing Changes to a Remote Repository:

`git push origin <branch-name>

12. Fetching Changes from a Remote Repository:

- `git fetch`

13. Creating a Tag:

- `git tag <tag-name>`

- `git tag -a <tag-name> -m "Tag message"`

14. Viewing Remote Repositories:

- `git remote -v`:

15. Undoing Changes:

git reset <filename>`

git reset --hard`

16. Creating a Pull Request (GitHub/GitLab):

- After pushing changes to a fork or branch, we can create a pull request on platforms like GitHub or GitLab to propose changes to the original repository.

3.GITHUB, GITLAB AND BITBUCKET

GitHub, GitLab, and Bitbucket are three popular web-based platforms that provide hosting and collaborative tools for version control using Git. GitHub, GitLab, and Bitbucket are platforms that leverage the Git version control system, providing additional features for collaboration, project management, and integration with other tools to enhance the development workflow. The choice among them often depends on individual or team preferences, specific project requirements, and integration needs.

GitHub:

GitHub is a web-based platform built around Git that provides a collaborative environment for hosting and managing Git repositories.

Key Features:

Repository Hosting: Host and share your Git repositories online.

Collaboration: Supports collaborative development with features like issue tracking, pull requests, and code review.

Social Coding: Users can follow projects, star repositories, and contribute to open source projects.

Integration: Integrates with various third-party services, such as continuous integration tools.

Public and Private Repositories: Provides options for both public and private repositories.

GitLab:

GitLab is a web-based platform similar to GitHub, offering Git repository hosting and collaboration features.

Key Features:

Integrated DevOps:GitLab provides a complete DevOps platform, including CI/CD (Continuous Integration/Continuous Deployment) tools.

Issue Tracking:Enables project management and issue tracking within the same platform.

Container Registry: Allows the storage and management of Docker images.

Built-in Wiki and CI/CD:Comes with a built-in wiki for documentation and robust CI/CD capabilities.

Public and Private Repositories: Like GitHub, GitLab supports both public and private repositories.

Bitbucket

Bitbucket is another web-based platform for hosting and managing Git repositories, provided by Atlassian.

Key Features:

Integration with Jira: Seamless integration with Jira for issue tracking and project management.

Mercurial Support: Besides Git, Bitbucket also supports the Mercurial version control system.

Code Review: Built-in tools for code review and collaboration.

Branch Permissions: Granular control over branch permissions for better access control.

Public and Private Repositories: Supports hosting of both public and private repositories.

4.INDUSTRIAL PRACTICES OF GIT

1. Version Control for Source Code:

- Git is used to track changes in source code, allowing developers to collaborate on projects and manage the entire history of code modifications.

- Different branches are often utilized for development, testing, and production releases to maintain a stable and organized codebase.

2. Collaborative Development:

- Teams of developers work concurrently on different aspects of a project using branches.

- Pull requests (or merge requests) are used for code reviews, allowing team members to review proposed changes before merging them into the main codebase.

3. Release Management:

- Git tags are often employed to mark specific releases or versions of the software.

- Branching strategies like GitFlow or GitHub Flow help manage the release process, ensuring a systematic approach to feature development, testing, and deployment.

4.Continuous Integration and Continuous Deployment (CI/CD):

- Git is integrated with CI/CD pipelines to automate the building, testing, and deployment of software.

- Changes pushed to specific branches trigger automated workflows, ensuring that the codebase is continuously integrated and validated.

5. Issue Tracking and Project Management:

- Many industrial teams use the issue tracking features provided by platforms like GitHub or GitLab to manage tasks, bugs, and feature requests.

- Integration with project management tools such as Jira helps maintain a comprehensive view of project progress.

6. Code Reviews:

- Git facilitates code reviews through pull requests or merge requests.

- Teams use comments, reviews, and approvals to ensure code quality, maintainability, and adherence to coding standards.

7. Git Hooks

- Git hooks are scripts that can be triggered before or after specific Git events (e.g., pre-commit, post-receive).

- Industrial practices might include custom hooks for enforcing coding standards, running tests, or triggering additional processes.

8. Branching Strategies:

Teams often adopt specific branching strategies, such as GitFlow or GitHub Flow, to manage feature development, hotfixes, and releases effectively.

- These strategies provide guidelines on how to create, merge, and handle branches throughout the development lifecycle.

9. Git Submodules:

- In large industrial projects, Git submodules might be used to include external repositories as dependencies within a main project.

- This helps manage complex project structures and dependencies.

10. Documentation:

- README files, wikis, or documentation folders in Git repositories are essential for conveying project information, setup instructions, and coding guidelines.

5.CLONING REPO TO LOCAL

Cloning a repository to your local machine is a common operation in Git, and it's a way to get a local copy of a remote repository.

1. Open a Terminal or Command Prompt

2. Navigate to the Directory Where You Want to Clone the Repository:

Use the `cd` command to change your current directory to the location where you want the cloned repository to be stored. For example:

```bash

cd path/to/your/directory

3. Clone the Repository:

Use the `git clone` command followed by the URL of the repository you want to clone. The URL can be obtained from the repository on the hosting platform (GitHub, GitLab, Bitbucket, etc.).

```bash

git clone <repository\_url>

```

For example, if you were cloning a repository from GitHub, the command might look like:

```bash

git clone https://github.com/example/repo.git

```

4. Provide Authentication if Required:

If the repository is private or requires authentication, Git might prompt you for a username and password, or you might need to use an access token. Enter the required credentials when prompted.

5. Wait for the Cloning Process to Complete:

Git will download the entire repository along with its history to your local machine. The progress will be displayed in the terminal.

6. Navigate into the Cloned Repository:

- Once the cloning process is complete, use the `cd` command to enter the newly cloned repository directory:

```bash

cd repo

6.RESOURCES

1. <https://docs.github.com/en/repositories/creating-and-managing-repositories/cloning-a-repository?tool=desktop>
2. <https://youtu.be/8JJ101D3knE?si=picEjMB6jp5l8xfF>
3. <https://www.atlassian.com/git/glossary#commands>
4. <https://www.atlassian.com/git/tutorials/git-bash>