Git documentation

1. What is Git?

Git is a distributed version control system (DVCS) designed to handle everything from small to very large projects with speed and efficiency. It allows multiple developers to collaborate on a project by tracking changes, managing versions, and enabling easy rollbacks if necessary.

2. Features of Git

- Distributed System: Every user has a full copy of the repository.

- Branching and Merging: Efficiently create, manage, and merge branches.

- Fast Performance: Most operations are performed locally, making Git extremely fast.

- Data Integrity: Ensures data integrity using SHA-1 hashing.

- Lightweight and Open Source: Git is free and has a small footprint.

- Collaboration: Easily work with remote repositories through platforms like GitHub, GitLab, and Bitbucket.

3. Types of Git

Centralized Version Control System (CVCS)

A Centralized Version Control System (CVCS) is a system where a single central server stores all the versions of a project, and multiple clients connect to this server to retrieve or update files.

Advantages:

- Simple to set up and use.

- All data is stored in a single location, making backup and maintenance easy.

- Ensures control over access, making it easy to enforce security.

Disadvantages:

- If the central server fails, all data could be lost or become inaccessible.

- Developers need a network connection to access version control.

- Slower operations due to reliance on a central server.

4.Distributed Version Control System (DVCS)

A Distributed Version Control System (DVCS) is a system where every developer has a full copy of the entire repository, including history, on their local machine. Git is an example of a DVCS.

Advantages:

- Works offline since each user has a local copy of the repository.

- Faster operations as many tasks (like commits, diffs, and logs) are performed locally.

- Improved redundancy and reliability, as data is not lost if the central server crashes.

- Facilitates better collaboration with branching and merging features.

Disadvantages:

- Requires more local storage since each user maintains a full copy of the repository.

- Initial cloning of a repository can be slow.

- Higher complexity compared to CVCS.

5. What is GitHub?

GitHub is a cloud-based platform that provides Git repository hosting along with collaboration tools. It allows developers to store, manage, and track changes in their code while working in teams. It offers features like:

- Pull Requests and Code Reviews
- Issue Tracking
- Continuous Integration/Continuous Deployment (CI/CD)
- Private and Public Repositories
- Integration with various DevOps tools

Use Case:

GitHub is used by open-source communities, organizations, and individual developers to manage projects and collaborate efficiently.

6.What is GitLab?

GitLab is a web-based DevOps platform that provides Git repository hosting along with an integrated set of CI/CD tools. Unlike GitHub, GitLab offers self-hosting options, making it suitable for enterprises that require full control over their repositories.

Key Features:

- Built-in CI/CD for automated testing and deployment

- Self-hosting options for on-premise installations

- Advanced security and permission management

- Agile project management tools

Use Case:

GitLab is used by enterprises that require private, self-hosted repositories with built-in CI/CD.

7. When to Use Git, GitHub, and GitLab?

Tool	When to Use?

Git	Use Git when working on local repositories for version control without needing cloud storage.
GitHub	Use GitHub when collaborating on open-source or private projects that require cloud hosting, pull requests, and issue tracking.
GitLab	Use GitLab when you need self-hosting capabilities and built-in CI/CD pipelines for automating deployments.

8. Differences Between Git, GitHub, and GitLab

Feature	Git	GitHub	GitLab
Type	Version Control System	Cloud-based Git hosting	Cloud-based and Self-hosted Git hosting
Hosting	No hosting	Cloud-based	Cloud-based & Self-hosted
Collaboration	No built-in tools	Supports collaboration with pull requests, issue tracking	Supports collaboration with more DevOps tools
CI/CD	Not available	Requires third-party tools	Built-in CI/CD
Security	Local control	Provides basic security features	Advanced security controls for enterprises

9. Installing Git

Windows:

Download Git from the official website: https://git-scm.com/downloads

Run the installer and follow the instructions. Ensure the following options are selected:

- Use Git from the Windows Command Prompt
- Use the OpenSSL library (for HTTPS authentication)
- Checkout Windows-style, commit Unix-style line endings

Click Install and wait for the process to finish.

Verify installation by opening Command Prompt and running: `git --version`

Linux (Ubuntu/Debian):

Open the terminal and run:

```
  sudo apt update

  sudo apt install git
```

Verify installation: `git --version`

MacOS:
Open Terminal and type: `brew install git`

(Make sure you have Homebrew installed: https://brew.sh)

Verify installation: `git --version`

10. Setting Up Git
Configure User Name and Email

```
  git config --global user.name "Your Name"

  git config --global user.email "your-email@example.com"
```

Verify configuration: `git config --list`

11. Initializing a Repository
Open a terminal or command prompt.

Navigate to your project folder: `cd /path/to/your/project`

Initialize Git in the project folder: `git init`

This creates a `.git` folder, which tracks changes.

12. Adding and Committing Files
To add a single file: `git add filename.txt`

To add all files in the directory: `git add .`

After adding files, commit them with a message: `git commit -m "Initial commit"`

13. Checking Git Status and Logs
Check the status of your repository: `git status`

View commit history: `git log`

14. Branching and Merging
```
git branch <branch-name>

git checkout <branch-name>
```

git merge <branch-name>

15. Undoing Changes

git reset --soft HEAD~1

git reset --hard HEAD~1

git revert <commit-hash