Git Introduction

What is Git and why is it used?

- Git is a version control system used to track changes in files and collaborate on software development projects.
- It helps developers manage changes to code, work together, and maintain project history.

Git types: 2types(CVCS, DVCS)

- 1. **Centralized Version Control System (CVCS)**
- A **Centralized Version Control System** (CVCS) uses a **central repository** where all versions of files are stored. Developers "check out" a copy of the code from this central repository, make changes, and then "check in" or "commit" those changes back to the central server.
- **Examples of CVCS:**
- **Subversion (SVN)**
- **CVS (Concurrent Versions System)**

Advantages of CVCS

- Simplicity and Ease of Use
- CVCS tends to be simpler and easier to set up and use compared to DVCS systems like Git, especially for beginners.
- Centralized Control and Management
- Since the version history is stored in one central repository, administrators have complete control over access and security. They can easily enforce policies on who can commit, access, or view the repository.
- **Easy to Set Up and Maintain**
- CVCS requires fewer resources and less complex configuration compared to a DVCS. There's only one central repository to set up and maintain.

Disadvantages of CVCS:

- **Single Point of Failure:** If the central server is down or unavailable, developers cannot commit, pull, or update code.
- **Limited Offline Work:** Developers can only work with the latest version of files and need to be connected to the server to commit changes.
- **No Complete History Locally:** The full version history is stored on the server, so developers can't easily browse or revert changes without connecting to the central repository.

DVCS:

- A **Distributed Version Control System** (DVCS) does not rely on a central server for its operations. Instead, every developer’s local machine is a full-fledged repository with its own history. Changes are **tracked and stored locally** in each developer’s machine, and they can commit, branch, and merge changes without needing constant access to a central server.
- **Examples of DVCS:**
- **Git**
- **Mercurial (Hg)**
- **Bazaar**

Advantages of DVCS

- **Work Offline:** Developers can work on the project without needing to be connected to the internet. They have the full project history locally.
- **No Single Point of Failure:** Even if the central repository is down, developers can continue working and then push their changes once the server is available.
- **Complete History:** Since every developer has the full history locally, they can view, branch, and revert any commit without needing access to a central server.
- **Easier Branching and Merging:** DVCS allows for easy branching and merging, encouraging workflows where features and experiments can be worked on independently and merged back easily.

- **Disadvantages of DVCS:**

- **More Storage:** Since each developer has a full copy of the repository, it may take up more storage space on their local machine.
- **Initial Setup Complexity:** The decentralized nature of DVCS can introduce more complexity, especially in setting up workflows or managing multiple repositories.

difference between Git and GitHub.

- Git:
- Git is a version control system that allows you to track changes in your code locally.

- GitHub:
- GitHub is a cloud-based platform that hosts Git repositories
- online, enabling collaboration and sharing of code.

How do you configure your username and email in Git?

You can configure your username and email by using the following commands:

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

git config --global user. Email "youremail@example.com" ex:

git config --global user.name "Suneetha"

git config --global user. Email "suneethavemula@puropalecreations.com"

How do you create a new Git repository?

- To create a new Git repository:
- Open the terminal/command line.
- Navigate to your project folder.
- Run the command: git init
- This will create a new Git repository in that directory.

How do you add files to the staging area in Git?

- To add files to the staging area, use the command: git add <filename>
- Or, to add all files:
- git add .
- This prepares the files to be committed.

How do you create a new commit in Git?

- Stage the changes by adding the modified files to the staging area using:
- git add <filename>   # Or use `git add .` to add all changes
- Commit the changes with a message that describes the changes:

- git commit -m "Your commit message"
- This will create a new commit that includes the staged changes and the message.

How do you view the changes made in a commit?

- To view the changes made in a specific commit, use the git show command followed by the commit hash:
- git show <commit-hash>
- This will display the changes (diff) made in that commit, including added or deleted lines in files.

How do you create a new branch in Git?

- To create a new branch in Git, use the following command:
- git branch <branch-name>
- This creates the new branch locally. To switch to the new branch after creating it, use:
- git checkout <branch-name>
- Alternatively, you can create and switch to a new branch in one step with:
- git checkout -b <branch-name>

Create a new Git repository and configure your username and email.

- Step 1: Initialize a new Git repository

- To create a new Git repository, follow these steps:

- Open your terminal/command prompt.

- Navigate to the directory where you want to create your project:
- cd /path/to/your/directory Initialize a new Git repository:
- git init

- Step 2: Configure your username and email

- After initializing the repository, you need to configure your username and email (this will be used for your commits):
- git config --global user.name "Your Name"

- git config --global user. Email "youremail@example.com" EX: git config --global user.name "Suneetha"
- git config --global user. Email "suneethavemula@puropalecreations.com"

SSH key Generation

- 1. **Open your terminal or command prompt**
- **Linux/macOS**: Open Terminal.
- **Windows**: You can use **PowerShell** or **Git Bash** (if Git is installed).
- 2. **Check for existing SSH keys**
- Before generating a new key, check if you already have existing SSH keys. Run the following command:
- `ls -al ~/.ssh`
- 3. **Generate a new SSH key pair**
- If you don't have an existing SSH key, or you want to create a new one, run the following command:
- bash
- Copy
- `ssh-keygen -t rsa -b 4096 -C "your_email@example.com"`
-
- **`-t rsa`**: Specifies the type of key to create (RSA in this case).
- **`-b 4096`**: Specifies the key size (4096 bits is recommended for strong security).
- **`-C "your_email@example.com"`**: This is a label for the key, typically your email address, which can help identify the key later.

- Example:
- `ssh-keygen -t rsa -b 4096 -C your_email@example.com`
- 4. **Choose a file to save the key**
- After running the above command, you’ll be prompted to specify the file location where the key will be saved:
- `Enter file in which to save the key (/home/):`
- 6. **Key generation complete**
- Once you've completed the previous steps, your SSH key pair will be generated. You should see a message like this:

```
Your identification has been saved in
/home/your_user/.ssh/id_rsa.

Your public key has been saved in
/home/youruser/.ssh/id_rsa.pub.

The key fingerprint is:

SHA256:xyz.... your_email@example.com
```

Create a file, add some content to it, and commit the changes.

- Step 1: Create a file and add some content
- Create a new file (for example, example.txt) and add
- some content to it:
- echo "This is my first commit!" > example.txt
- Step 2: Add the file to the staging area
- After creating the file, add it to Git's staging area: git add example.txt
- Step 3: Commit the changes
- Now, commit the changes with a descriptive message: git commit -m "Added example.txt with initial content"

Clone an existing repository from GitHub and make some changes.

- Step 1: Clone the repository from GitHub
- To clone an existing repository, you need the URL of the repository. Go to the repository's GitHub page and copy the URL.
- Then, run the following command in your terminal to clone it:
- git clone https://github.com/username/repository- name.git
- Replace https://github.com/username/repository- name.git with the actual repository URL.
- Step 2: Make some changes
- After cloning, navigate into the cloned repository's directory:
- cd repository-name
- Make some changes to a file or create a new one. For example, you can add a new file:
- echo "Some changes made" > changes.txt

- Step 3: Add, commit, and push changes
- Add the changes to the staging area: git add changes.txt
- Commit the changes:

- git commit -m "Added changes.txt with some content"

- Push the changes to the GitHub repository:

- git push origin main

Create a new branch, make some changes, and switch back to the main branch.

- Step 1: Create a new branch
- To create a new branch, use the following command: git branch new-branch
- This creates a new branch named new-branch but does not switch to it. To switch to the new branch, use:
- git checkout new-branch
- Alternatively, you can use this single command to create and switch to the new branch:
- git checkout -b new-branch

- Step 2: Make changes in the new branch
- Once you’re on the new branch, make some changes (e.g., create or modify a file):
- echo "Changes on new branch" > new-file.txt

- Add and commit the changes:
- git add new-file.txt
- git commit -m "Created new-file.txt on new-branch"

Step 3: Switch back to the main branch

- Step 3: Switch back to the main branch
- After committing your changes in the new branch, you can switch back to the main branch:
- git checkout main
- Alternatively, if you're using Git 2.23 or later, you can use git switch:
- git switch main
- Now you’re back on the main branch. If you want to merge the changes from new-branch into main, you can do so by running:
- git merge new-branch

What is vcs(version control system)

- a **Version Control System (VCS)** is a tool that helps you **track and manage changes** to files, typically in a software project. It allows you to save different versions of a file or project over time, so you can **revert back to previous versions** if needed, **collaborate with others**, and keep a clear history of all the changes made.
- Examples:
- **Git**: A popular VCS used by many developers, especially for software projects.
- **SVN (Subversion)**: Another VCS used to manage code changes in teams.

2. What is Collaboration?

- **Collaboration** refers to the process of working together with others towards a common goal, often involving multiple people. In software development, collaboration involves **communicating, sharing ideas, dividing tasks**, and **integrating contributions** from multiple developers or teams to create a cohesive product.
- In the context of software development, **collaboration** is not limited to just code — it encompasses all aspects of working together, including:
- **Design** and architecture
- **Issue tracking** and task management
- **Code reviews** and discussions
- **Documentation** and knowledge sharing
- **Continuous Integration / Continuous Deployment (CI/CD)**

Functionalities of vcs

- **1. Tracking Changes to Files**
- **What It Does**: A VCS records every change made to files, such as added, modified, or deleted content. It keeps a record of who made the change, when it was made, and why (through commit messages).
- **2. Managing Versions of Files**
- **What It Does**: A VCS lets you save snapshots (versions) of your project at different points in time. This means you can revert to any prior version of a file or the entire project whenever needed.
- **3. Collaboration and Concurrent Editing**
- **What It Does**: A VCS allows multiple developers to work on the same project simultaneously. It provides mechanisms to **merge changes** and resolve **conflicts** if two or more developers make changes to the same part of a file.

Difference of SVN and Git

Feature	CVCS (e.g., SVN)	DVCS (e.g., Git)
Repository	Centralized, single server	Distributed, each developer has a full repo
Local History	No (only the latest working copy)	Yes (full history locally)
Offline Work	Cannot commit without the central server	Can commit, branch, and work offline
Network Dependency	Always needs network access to commit	No, except for pushing/pulling changes
Branching and Merging	More difficult and error-prone	Very flexible and efficient
Examples	SVN, CVS, Perforce	Git, Mercurial, Bazaar

What is protocols

- a **protocol** refers to the communication method used to transfer data between your local machine and the **remote repository**. When interacting with a remote repository .
- 1. **HTTPS (HyperText Transfer Protocol Secure)**
- **Protocol**: `https://`
- The **HTTPS** protocol is the most common and widely used method to communicate with remote Git repositories. It uses **SSL/TLS encryption** to secure the communication between the client (your machine) and the server (remote repository).
- HTTPS is often preferred for its simplicity, security, and ease of use, especially when interacting with services like **GitHub**, **GitLab**, or **Bitbucket**.

SSH(Secure Shell)

- **Protocol**: `ssh://`

SSH is a secure network protocol used to access remote machines. Git can use SSH to communicate with a remote repository over a secure, encrypted channel, and it's often used by developers who want a more seamless authentication experience.

SSH is highly recommended for users who want to **automate authentication** (i.e., no need to enter a username/password every time) and prefer not to use HTTPS tokens.

Git commands?

1. **`git version`**

- Displays the current version of Git installed on your system.
- To check which version of Git is installed.

```
git –version
```

2. **`git config`**

- Used to set Git configuration options such as user information (name, email) and other preferences for how Git behaves.

Set your **name** and **email** for commits.

```
git config --global user.name "Your Name"
git config --global user.Email "your.email@example.com"
```

- **3. `git init`**
- Initializes a new Git repository in your current directory. It creates a `.git` folder that stores all the metadata and version history.
- Run this command when you want to start tracking a new project with Git.
- `git init`
- **4. `git commit`**
- Records changes to the repository, with a message describing the changes made. Each commit has a unique ID.
- Committing staged changes to the local repository.
- Example:
- `git commit -m "Initial commit"`
- **`-m "message"`**: Adds a commit message explaining the changes.

- **5. `git remote`**
- Manages remote repositories. A remote repository is a version of your project hosted on a server (like GitHub, GitLab, etc.).
- Add, remove, or view remote repositories.
- View current remotes:
- `git remote -v`
- Add a remote repository (usually used to connect a local repo to GitHub, GitLab, etc.):
- bash
- Copy
- `git remote add origin` https://github.com/username/repository.git

- **6. `git push`**
- Uploads local commits to a remote repository, like pushing your code changes to GitHub.
- Push local commits to a remote repository to make them available to other collaborators.
- Example:
- `git push origin main`
-
- `origin`: The name of the remote (usually `origin`).
- `main`: The branch you want to push to (could be `master`, `main`, or any other branch).

- **7. `git add`**
- Stages changes (new, modified, or deleted files) for the next commit. You need to **add** files before you can **commit** them.
- Used after making changes to files to prepare them for committing.
- Stage a specific file:
- `git add filename.txt`
-
- Stage all changed files:
- `git add .`
-
- `.` stages all changes in the current directory and subdirectories.

- **git status**
  Displays the status of the working directory and staging area.
- **git log**
  Shows the commit history.

```
git log
```

- **git diff**
  Displays the differences between files or commits.

```
git diff  # Shows changes between working directory and staged files
```

git rm
Removes files from the working directory and stages the removal.

```
git rm filename.txt
```

- **3. Branching and Merging**
- **`git branch`**
  `Lists all branches or creates a new branch.`
  - List all branches:
  - `git branch`
  -
  - Create a new branch:
  - `git branch new-branch`
- Create and switch to a new branch:
  - `git checkout -b new-branch`
- **`git merge`**
  `Merges changes from one branch into the current branch.`
  - `git merge branch-name`

- **`git rebase`**
  Reapplies commits from one branch onto another (use carefully to avoid rewriting history).

`git rebase branch-name`

- **4. Remote Repositories**
- **`git remote`**
  Manages remote repositories.
  - List remotes:
  - `git remote -v`
  -
  - Add a remote:
  - `git remote add origin` https://github.com/user/repository.git

- **git push**
  Pushes local commits to a remote repository.
- git push origin branch-name
-
- **git pull**
  Fetches and merges changes from the remote repository.
- git pull origin branch-name
-
- **git fetch**
  Fetches changes from the remote repository without merging them.
- git fetch
-
- **git clone**
  Clones a remote repository to your local machine.
- git clone https://github.com/user/repository.git

- **git checkout --**
-
  Discards changes in a specific file.
- `git checkout -- filename.txt`
-
- **git revert**
-
  Reverts a commit by creating a new commit that undoes the changes.
- `git revert <commit-id>`
-
- **6. Viewing History and Logs**
- **git log**
-
  Shows commit history.
- `git log`
-
- **git log –oneline**
-
  Shows a simplified commit history.
- `git log --oneline`
-
- **git show <commit-id>**
-
  Displays information about a specific commit.
- `git show 123abc`