What is Git?

Git is a distributed version control system designed to track changes in source code during software development. It enables multiple developers to work on a project simultaneously without conflicting with each other's work. Git allows users to maintain a complete history of file changes and revert to previous versions when needed.

Why do we use Git?

- **Collaboration:** Git allows multiple people to work on a project at the same time without overwriting each other's changes.

- **Version Control:** It keeps a history of changes, so you can track what was modified and when. It also allows you to roll back to earlier versions.
- **Branching and Merging:** Developers can create branches to work on different features or bug fixes, merge them back into the main codebase once completed, or even discard them if not needed.
- **Distributed Development:** Each developer has a full copy of the repository, making it possible to work offline and sync changes later.

Advantages of Git:

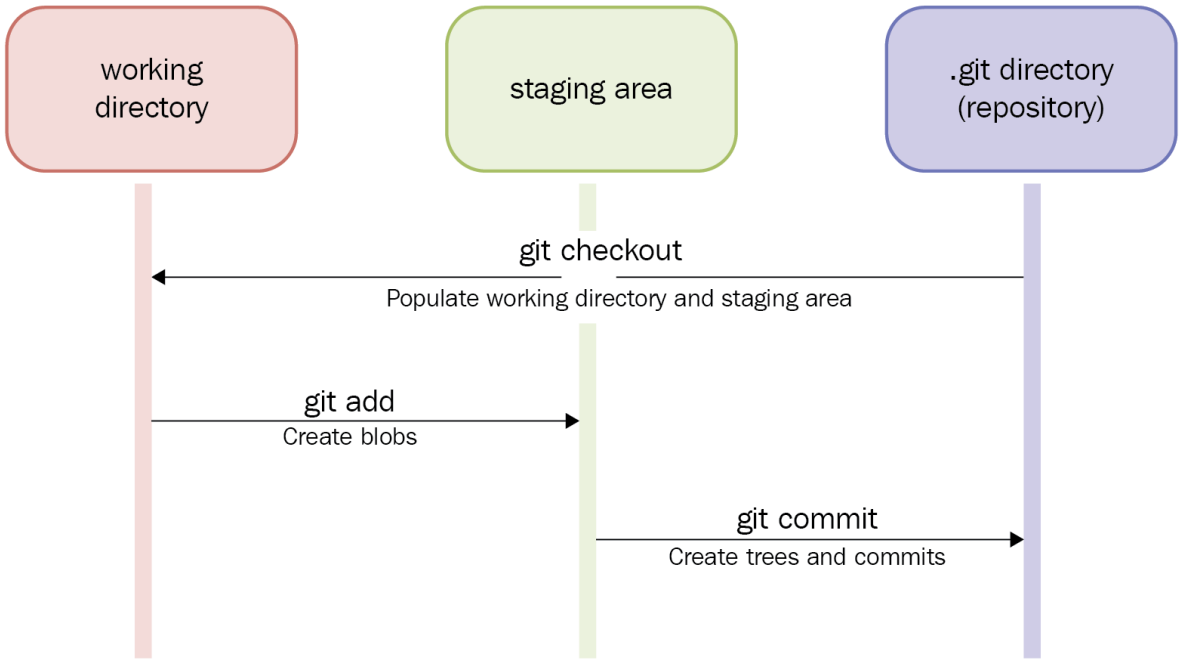
1. **Distributed Version Control:** Every user has a local copy of the entire project history, which makes it faster and more reliable, even when working offline.
2. **Efficient Branching and Merging:** Git makes it easy to create, switch, and merge branches, facilitating parallel development.
3. **Lightweight Operations:** Git operations are typically fast, as most actions are performed locally, reducing the need for network communication.
4. **Open Source and Free:** Git is freely available and can be used across different platforms.

Disadvantages of Git:

1. **Learning Curve:** Git can be complex for beginners due to the vast number of commands and workflows.

2. **Conflicts Handling:** While Git helps in avoiding conflicts, merging large projects or frequent branches may still lead to merge conflicts that require manual resolution.
3. **Disk Space Usage:** Since Git maintains the entire project history locally, it can take up a lot of space, especially in large repositories.

Git Structure:



1. **Repository:** A Git repository is where the project files and history are stored.
2. **Branch:** A branch is a separate version of the project, allowing developers to work independently.
3. **Commit:** A commit represents a snapshot of the project at a specific point in time.
4. **Clone:** A clone is a copy of the repository that can be worked on locally.

5. **Pull and Push:** Pulling fetches changes from a remote repository, while pushing uploads local changes to a remote repository.

6. **1. Repository (Repo):**
   A repository is like a folder where your project's files and their entire history of changes are stored. It’s where Git tracks everything.
7. **2. Branch:**
   A branch is like a separate line of work in your project. You can create branches to try out new things without affecting the main project.
8. **3. Commit:**
   A commit is a saved snapshot of your project. It records changes you’ve made at a certain point in time, so you can look back or undo them later.
9. **4. Clone:**
   Cloning is when you make a full copy of a project from an online repository to your own computer, so you can work on it.
10. **5. Pull:**
    Pulling is when you bring the latest changes from the remote (online) repository to your local repository on your computer.
11. **6. Push:**
    Pushing is when you send the changes you made on your computer back to the remote repository, so others can see and use them.
12. **7. Merge:**
    Merging combines changes from different branches back into the main branch or another branch, bringing everyone's work together.

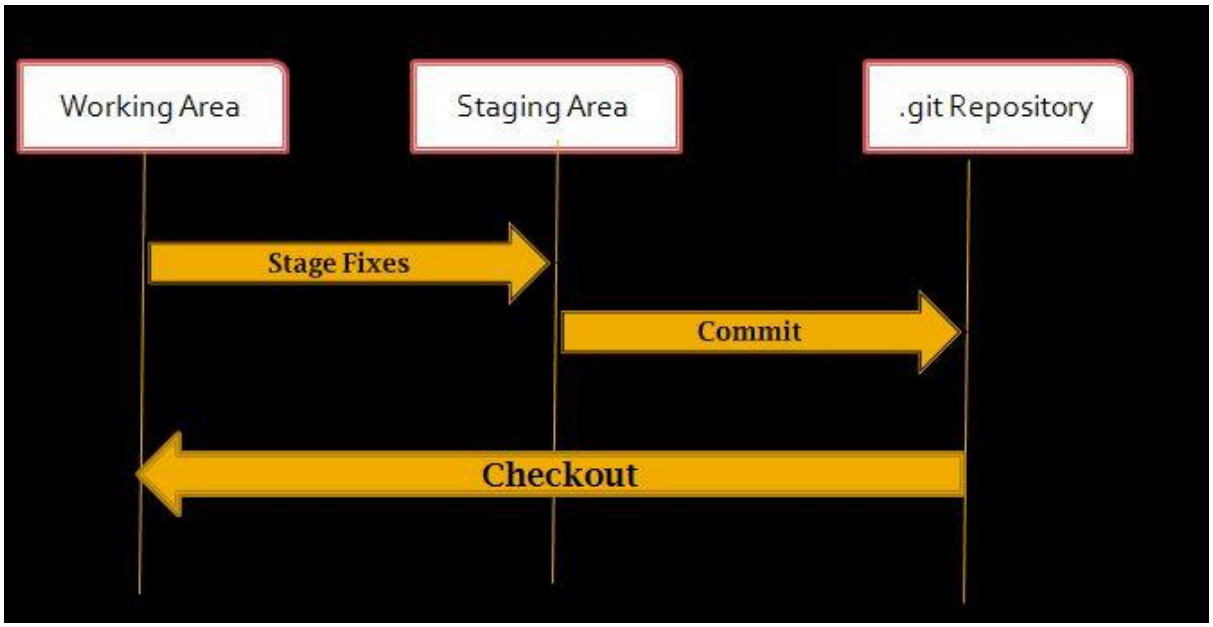
13. **8. Conflict:**
    A conflict happens when two people change the same part of a file in different ways, and Git doesn't know which version to keep. You have to decide which changes to keep.
14. **9. Staging Area:**
    The staging area is like a waiting room where changes sit before they are committed. You add files here before saving them with a commit.
15. **10. Checkout:**
    Checking out means switching to a different branch or a specific commit to work on or see how things were at that point.
16. **11. Remote Repository:**
    This is an online version of your repository, hosted on platforms like GitHub or GitLab. It's where everyone shares their work.

comparison between **DVCS (Distributed Version Control System)** and **CVCS (Centralized Version Control System)**:

CVCS (Centralized Version Control System)	**CVCS (Centralized Version Control System)**
Each user has a complete copy of the entire repository locally.	A single central server stores the repository; users get only snapshots.
Can work completely offline, as the full repository is on the local machine.	Requires connection to the central server to access the latest project updates.

Faster, since most operations (commits, branches) happen locally.	Slower, as most operations depend on the central server.
No single point of failure, as every user has a full backup of the repository.	The central server is a single point of failure; if it goes down, work is halted.
Easier branching and merging; multiple users can work on different parts independently.	Harder to manage parallel development; branches and merges are more complex.
Every user has the full history, so it's harder to lose data.	Security is managed centrally, but if the central server is compromised, everything is at risk.

The three stages of Git are:



1. **Working Directory (Workspace):**
   This is where you create, edit, or delete files. It's your local workspace where you make changes to the project.
2. **Staging Area (Index):**
   This is like a waiting area where you prepare the changes you want to commit. You add files here before making a commit. It allows you to review what will be included in the next commit.
3. **Git Repository (Committed):**
   Once changes are committed, they are stored in the repository. This is where the full history of your project is kept, including all changes and versions.

1. Installation Commands:

- **Install Git:**
  - **For Windows:**
    Download and install Git from the official website: https://git-scm.com/.
  - **For Linux:**

sudo apt-get install git

  - **For macOS:**

brew install git

2. Configuration Commands:

- **Set Username:**

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

This command sets the username for all repositories on your system.

- **Set Email:**

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

This sets the email associated with your commits.

- **Check Configurations:**

git config --list

Displays all Git configuration settings.

3. Repository Commands:

- **Initialize a New Repository:**

git init

Creates a new empty Git repository in your project folder.

- **Clone a Repository:**

git clone <repository-url>

Creates a local copy of a remote repository on your machine.

4. Basic File Operations:

- **Check the Status of Files:**

git status

Shows which files are in the working directory, staged, or ready to commit.

- **Add Files to Staging Area:**

git add <file-name>

Moves files from the working directory to the staging area (preparing them for commit).

Add All Files:

git add .

Adds all changes to the staging area.

- **Commit Changes:**

git commit -m "Your commit message"

Commits the staged changes with a descriptive message.

- **Remove Files:**

git rm <file-name>

Removes a file from the repository and stages the deletion.

5. Branching and Merging:

- **Create a New Branch:**

git branch <branch-name>

Creates a new branch.

- **Switch to a Branch:**

git checkout <branch-name>

Switches to an existing branch.

- **Create and Switch to a New Branch:**

git checkout -b <branch-name>

Creates and switches to the new branch in one command.

- **Merge Branches:**

git merge <branch-name>

Merges the changes from the specified branch into the current branch.

- **Delete a Branch:**

git branch -d <branch-name>

Deletes the specified branch.

6. Remote Repository Commands:

- **Add a Remote Repository:**

git remote add origin <repository-url>

Links your local repository to a remote repository (often named "origin").

- **View Remote Repositories:**

git remote -v

Lists all the remote repositories linked to your local repository.

- **Fetch Changes from Remote:**

git fetch

Fetches changes from the remote repository without merging them into your current branch.

- **Pull Changes from Remote:**

git pull

Fetches and merges changes from the remote repository into your current branch.

- **Push Changes to Remote:**

git push

Sends your local commits to the remote repository.

7. Tracking and Viewing History:

- **View Commit History:**

git log

Shows a list of all commits made in the repository.

- **View a Specific Commit:**

git show <commit-hash>

Shows details of a specific commit, including changes and metadata.

- **View Changes (Difference):**

git diff

Shows differences between your working directory and the staging area.

View Changes Between Branches:

git diff <branch1> <branch2>

Compares two branches.

8. Undoing Changes:

- **Unstage Files:**

git reset <file-name>

Removes a file from the staging area (but keeps the changes in the working directory).

- **Undo Last Commit (Keep Changes):**

git reset --soft HEAD~1

Removes the last commit but keeps the changes in the staging area.

- **Undo Last Commit (Discard Changes):**

git reset --hard HEAD~1

Removes the last commit and discards the changes entirely.

- **Revert a Commit:**

git revert <commit-hash>

Creates a new commit that undoes the changes made by a specific commit.

9. Stashing Changes:

- **Stash Changes:**

git stash

Temporarily saves your changes without committing them, allowing you to switch branches.

- **Apply Stashed Changes:**

git stash apply

Restores stashed changes to your working directory.

- **List Stashes:**

git stash list

Shows a list of stashed changes.

- **Drop Stash:**

git stash drop

Deletes a specific stash from the stash list.

10. Tagging:

- **Create a Tag:**

git tag <tag-name>

Tags a specific commit with a version number or label.

- **List All Tags:**

git tag

Shows all tags in the repository.

- **Push Tags to Remote:**

git push origin --tags

Pushes all tags to the remote repository.

11. Git Ignore:

- **Create a .gitignore File:** Inside this file, list the files or directories that Git should ignore. Example:

/node_modules

.env

Git will now ignore these files in the repository.

12. Collaboration Commands:

- **Fork a Repository:**
  Forking is done via platforms like GitHub or GitLab, where you copy someone's repository to your account to work independently.
- **Submit Pull Request:**
  After forking and making changes, you can submit a pull request via GitHub to propose your changes to the original repository.