Pimpri Chinchwad Education Trust's
Pimpri Chinchwad College of Engineering
An Autonomous Institute
(Permanently affiliated to Savitribai Phule Pune University)

BIT26VS01 - DevOps Laboratory

Name: Harshvardhan Atul Borude

PRN: 123B1F013

Title

Git Installation & Setup
a. Install Git on your system.
b. Configure Git with your name and email.
c. Check Git version and setup verification.

Objectives

To install Git and configure it with username and mail ID for version control usage

Theory

1. Git - Version Control System

The Role of Version Control in Modern Development

In any software engineering lifecycle, managing code evolution is just as critical as writing the code itself. **Version Control Systems (VCS)** act as the administrative backbone of a project, ensuring that every iteration is documented and every contribution is synchronized.

1. The Core Philosophy of Git

Git operates as a **Content Tracker**, but its true value lies in its ability to manage the "life story" of a codebase. Instead of saving a file multiple times with names like project_v1 or project_final, Git creates a linear or branched history that allows for:

- **Granular Accountability:** Every single line of code is linked to a specific author and a timestamp, providing a transparent audit trail.
- **Safety Nets (Rollbacks):** Development is inherently risky; Git removes the fear of failure by allowing developers to restore the entire system to a known stable state with a single command.
- **Non-Linear Development:** Through **Branching**, teams can diverge from the main project to experiment or build features in isolation, ensuring the production-ready code remains untouched until the new work is verified.

Why Version Control is Non-Negotiable

Without a robust VCS, collaboration becomes a series of manual conflicts. Version control solves three fundamental engineering challenges:

- **Systematic Change Management**

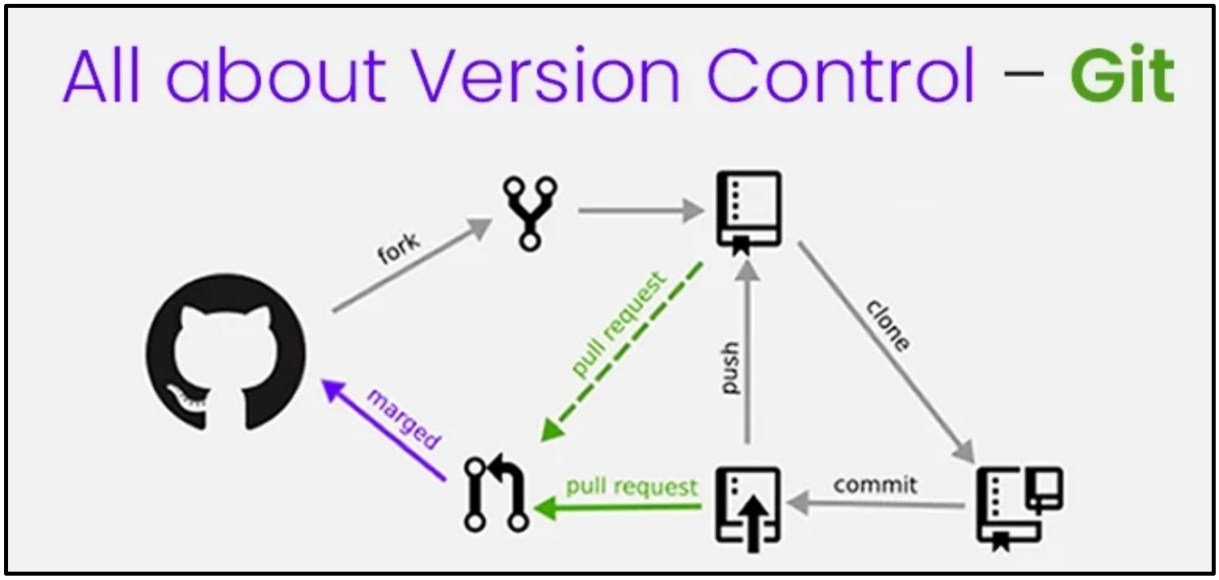
  Rather than just "saving" files, version control "commits" them. This creates a permanent record of the project's state. If a bug is introduced in a new update, developers can perform a "diff" to see exactly what changed between the working version and the broken one.

- **Conflict Resolution & Collaboration**

  When multiple engineers work on the same module, a VCS acts as the mediator. It identifies where changes overlap and provides tools to merge these contributions seamlessly, preventing the "last-save-wins" disaster common in shared folders.

- **Environment Consistency**

  By using a VCS like Git, every developer on a team has an identical copy of the project history. This ensures that the code running on a lead architect's machine is the exact same code being tested by a QA engineer.



2. Git Workflow

A Git workflow is a defined set of rules and practices that a development team agrees upon for using Git to manage code changes, collaborate effectively, and maintain a clean project history.
The Core Git Workflow Concepts
At its core, Git operates around a three-stage architecture that dictates how changes are tracked:

- Working Directory: Your local workspace where you actively edit, add, or delete files.
- Staging Area (Index): A temporary area where you prepare changes to be included in the next commit. This allows you to selectively choose which modifications to save together.

- Local Repository (.git directory): Where your committed changes are permanently stored as part of the project's history on your local machine.
- Remote Repository: A shared repository on a server (like GitHub or Bitbucket) that enables team collaboration and acts as the project's single source of truth.

Common Git Commands in the Workflow

Developers use a set of core commands to move changes through these stages:

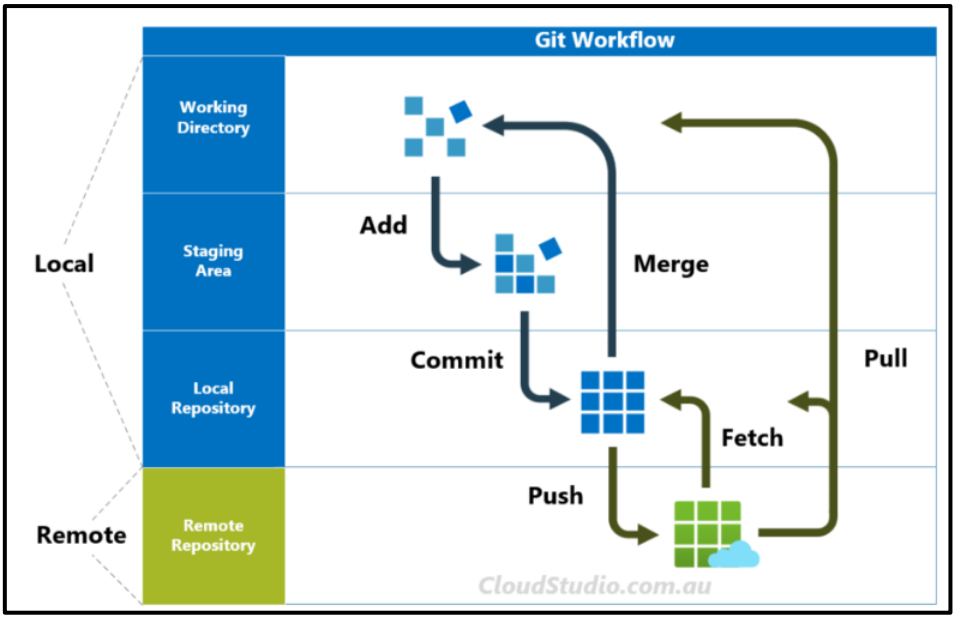
- git add <file>: Stages changes from the working directory to the staging area.
- git commit -m "message": Permanently saves the staged changes as a new commit in the local repository.
- git push: Uploads local commits to the remote repository, sharing them with the team.
- git pull: Fetches the latest changes from the remote repository and integrates them into your local branch, ensuring your local copy is up to date.
- git status: Checks the state of your working directory and staging area, showing which files are modified, staged, or untracked.

Popular Git Workflows and Branching Strategies

Different teams adopt different branching strategies (workflows) based on project needs and release schedules.

- Centralized Workflow: The simplest model, similar to SVN. All developers work on the main branch, pushing changes to a central repository. This is best for small teams or those transitioning to Git.
- Feature Branch Workflow: A logical extension where all new feature development happens in dedicated branches branched off main (or develop). This isolates work and ensures the main branch remains stable. Changes are integrated via pull requests and code reviews before merging back into the main branch.
- Trunk-Based Development (TBD): Prioritizes speed and continuous integration by using short-lived feature branches that are merged into a single main branch frequently, often multiple times a day. This approach relies heavily on automation and testing.
- Gitflow Workflow: A more complex, structured approach with dedicated branches for different stages: main (production-ready code), develop (integration of features), feature branches, release branches (for stabilization), and hotfix branches (for urgent production issues). This is suitable for projects with defined, scheduled release cycles.
- Forking Workflow: Commonly used in open-source projects. Each contributor has their own server-side copy ("fork") of the project. Changes are made in the fork and then submitted to the original repository's maintainers via a pull request for review and integration.

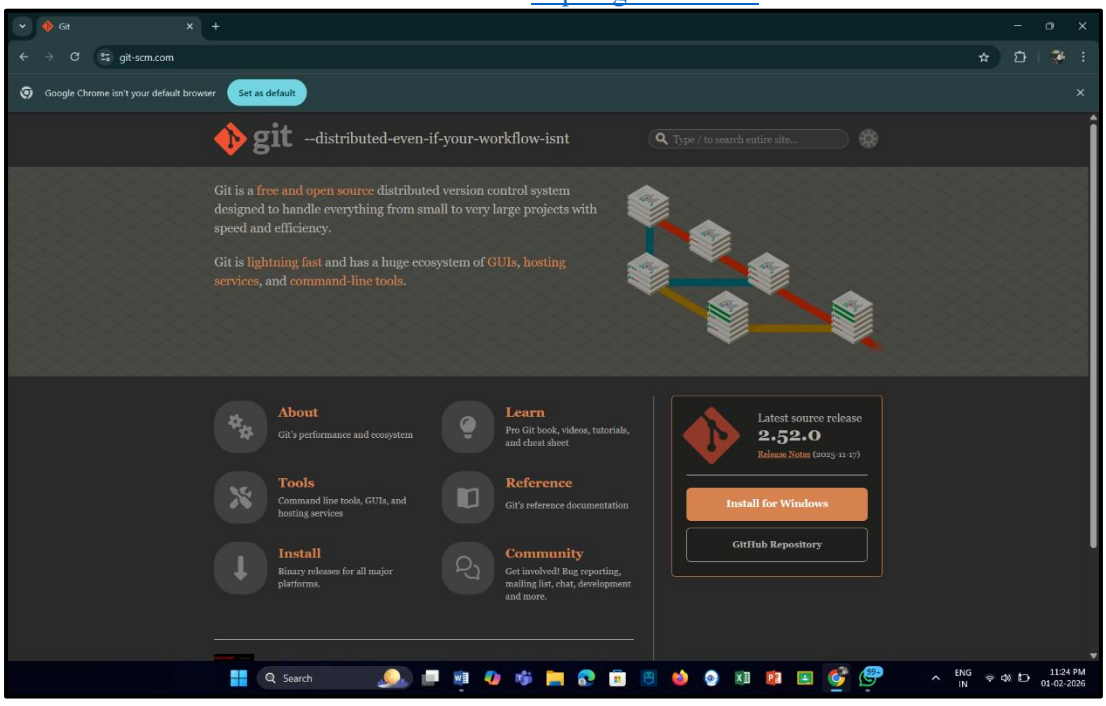
Choosing the right workflow depends on your team's size, culture, and delivery requirements. A successful workflow scales with the team, makes it easy to undo mistakes, and maintains code stability.

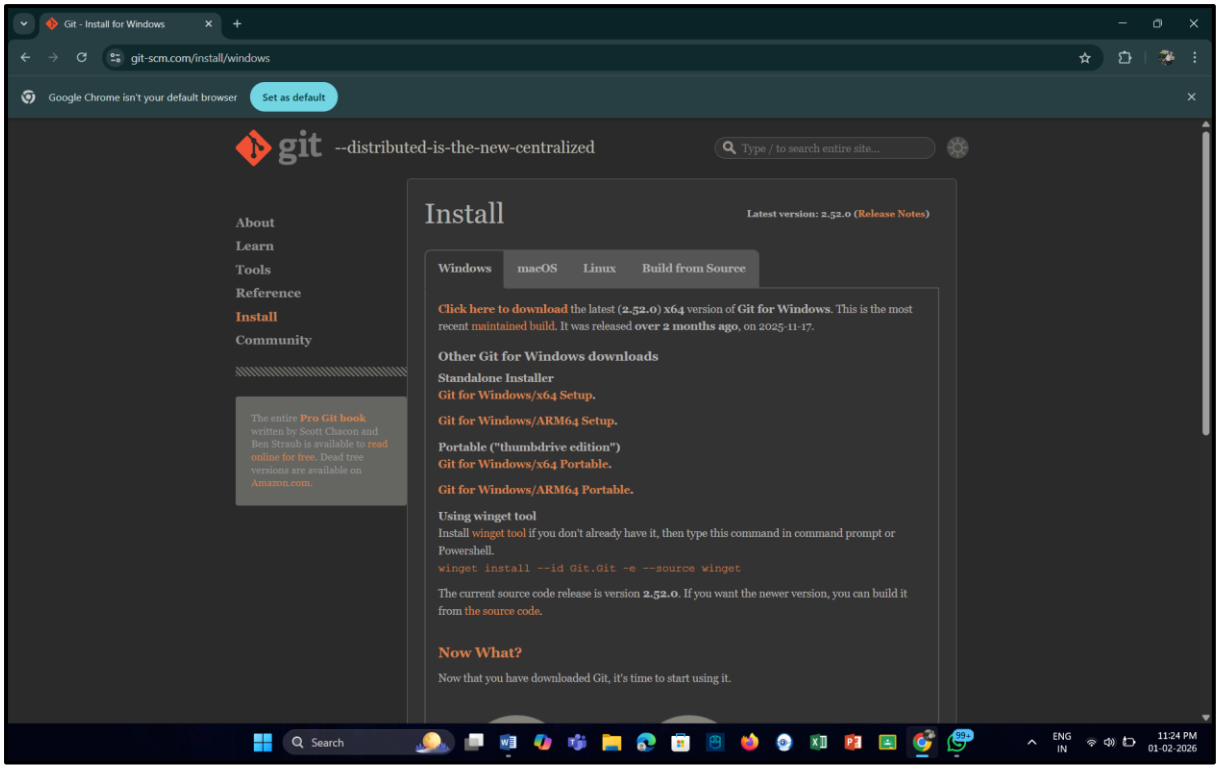


3. Git installation

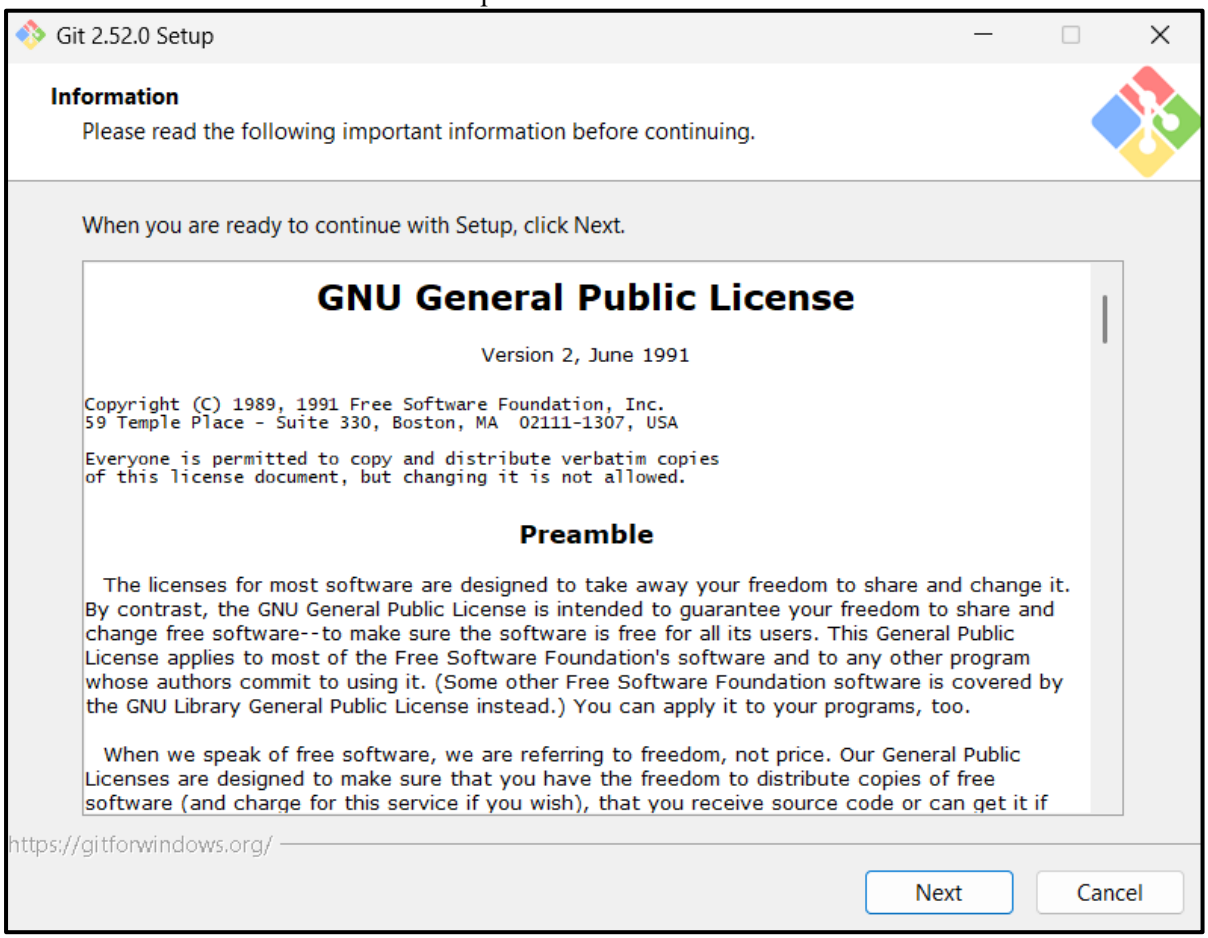
Steps to Install Git:

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

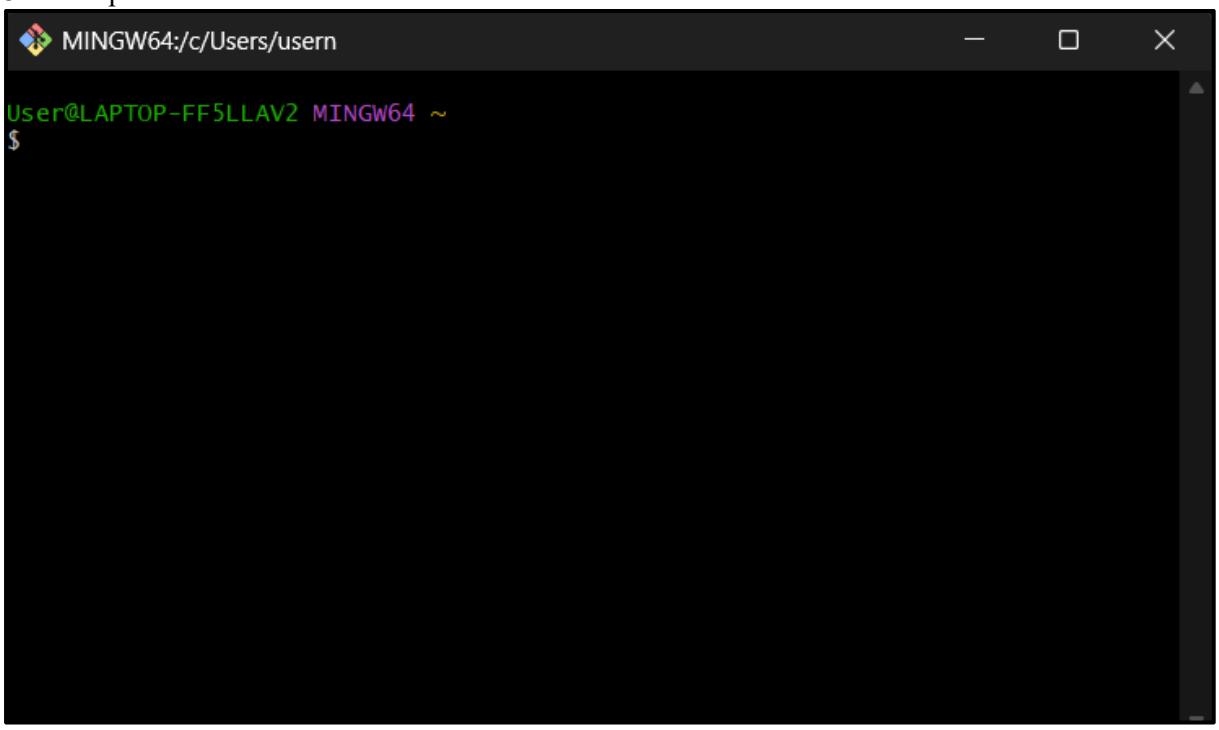




2. Run the installer and select default options

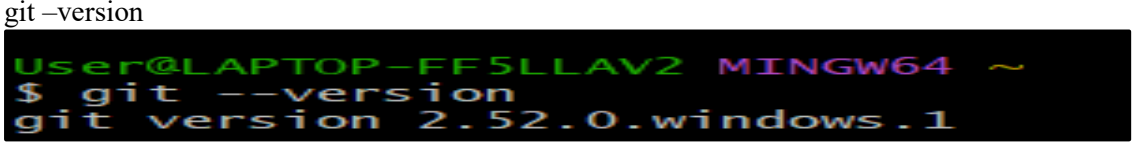


3. Complete installation

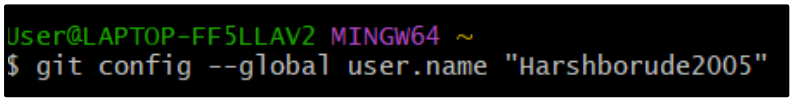


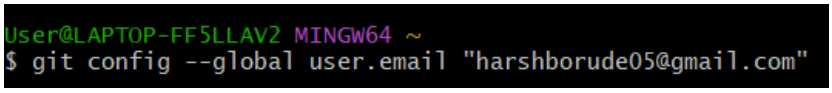
Verify Installation:

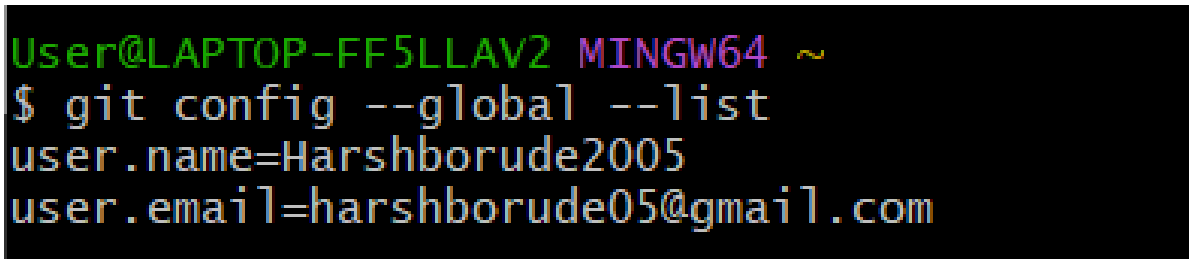
git –version



4. Configure Git with name and mail ID







FAQs

1. Difference between Git and GitHub.

Feature	Git	GitHub
What is it?	A local software (Version Control System).	A cloud-based web platform.
Where does it live?	On your computer (Local).	On a server (Remote).
Purpose	Manages history and versions of code.	Hosts Git repositories and facilitates collaboration.
Interface	Command Line (usually) or Desktop App.	Web-based Graphical User Interface (GUI).
Competition	Mercurial, Subversion (SVN).	GitLab, Bitbucket.

2. Explain the type of Version Control System.

Types of Version Control Systems
There are three main types of Version Control Systems:
1. Local Version Control Systems (Local VCS)
A Local Version Control System operates entirely on your personal machine without any connection to a remote repository. All changes and version history are stored in a local database on your computer.
In this setup, there is only a single user, with no collaboration or sharing of changes. Local VCS stores versions in a local database, not in a repository that others can clone.
Characteristics:

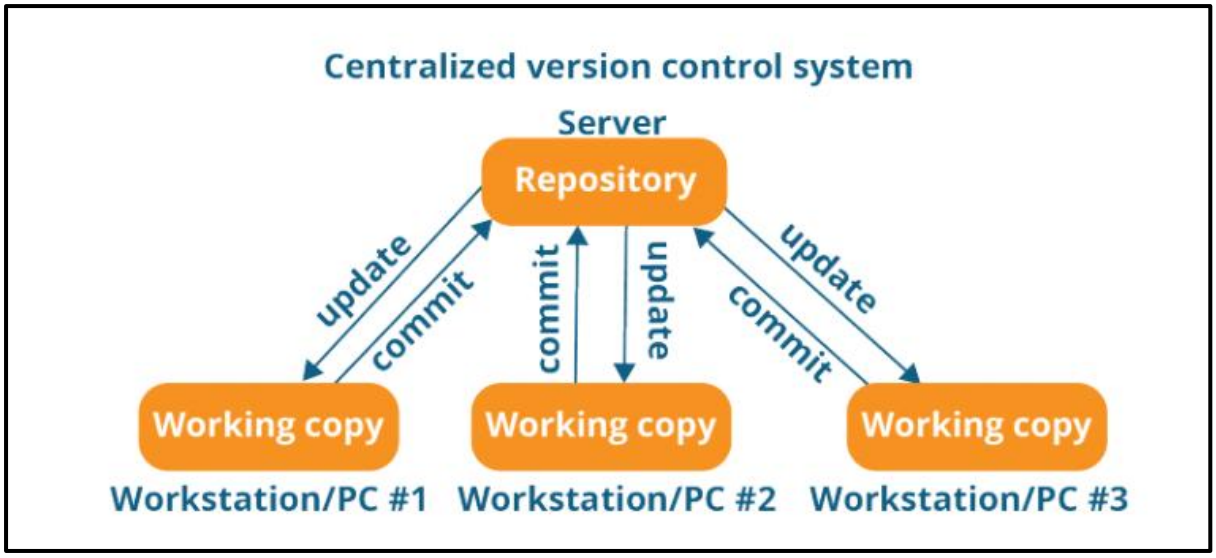
- No internet or server dependency.
- Useful for individual projects.
- Limited to single-user environments.

2. Centralized Version Control Systems
In a Centralized Version Control System, all the files and their version history are stored in a single central server. Developers connect to this server to access or modify files.
The typical workflow is:

1. Update/Checkout: A developer pulls the latest version of the files from the central server.
2. Make Changes: A developer works on the files.
3. Commit: A developer saves ("commits") their changes directly back to the central server, making them immediately available to everyone else.

Pros & Cons:

- The benefit of CVCS (Centralized Version Control Systems) is that it makes collaboration among developers while providing insight into what everyone else is doing on the project.
- It allows administrators to have fine-grained control over who can do what. It has some downsides as well which led to the development of DVCS.
- The most obvious drawback is the single point of failure represented by the centralized repository. If the repository goes down, you would not be able to collaborate or save changes.



3. Distributed Version Control Systems

Distributed version control systems contain multiple repositories. Each user has his or her own repository and working copy. Just committing your changes will not give others access to your changes. This is because a commit will reflect those changes in your local repository and you need to push them in order to make them visible to the central repository.

Similarly, When you update, you do not get others changes unless you have first pulled those changes into your repository.

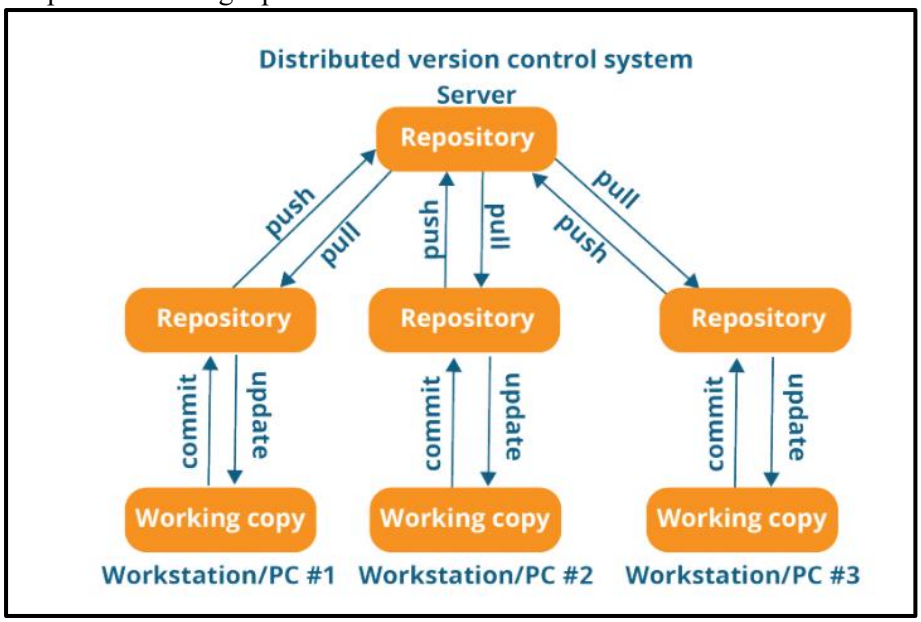
The key difference is the two-step process for sharing changes:

1. Commit: You save your changes to your own local repository. At this point, the changes are only on your machine; no one else can see them.
2. Push: You upload ("push") your committed changes from your local repository to the central repository (e.g., GitHub).

To get changes from others, the process is:

1. Fetch/Pull: You download ("pull") the latest changes from the central repository to your local repository.

The most popular distributed version control systems are Git and Mercurial. They help us overcome the problem of single point of failure.



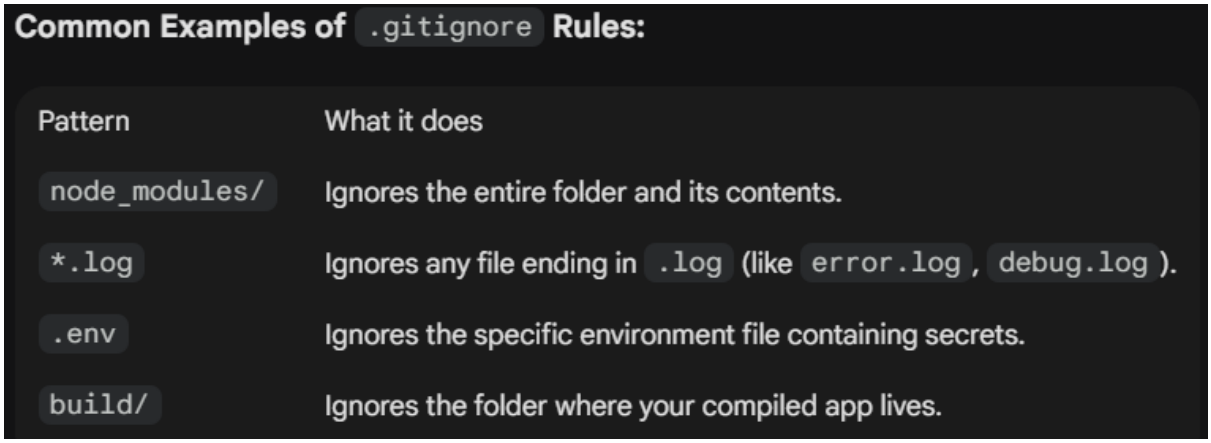
3. What is . gitignore?

In simple terms, .gitignore is a plain text file that tells Git **"don't look at these files."** When you are working on a project—especially in an IT lab or a real-world software job—your folder gets cluttered with files that don't belong in your permanent code history. The .gitignore file ensures these files stay on your local computer and never get uploaded to your GitHub repository.

Why do we need it ?

If you don't use a .gitignore, your repository becomes bloated and potentially dangerous. It helps you avoid:

- **Security Risks:** Accidentally uploading files that contain passwords, API keys, or private credentials (like .env files).
- **System Clutter:** Uploading OS-specific files that others don't need (like Mac's .DS_Store or Windows' Thumbs.db).
- **Dependency Bloat:** Uploading massive folders of third-party code (like node_modules in JavaScript) that can easily be re-downloaded by others using a package manager.
- **Build Artifacts:** Uploading compiled code (like .class files in Java or .exe in C) that are generated automatically when you run your program.



Conclusion:

Through this assignment, I successfully completed the initial environment setup required for modern version control. By installing Git and configuring the global user identity, I learned that version control is not just about saving files, but about establishing an **identity-linked history** of a project's evolution. The transition from understanding simple file backups to a **Distributed Version Control System (DVCS)** highlights the efficiency of Git, where every local machine holds a full copy of the repository's history. This setup serves as the foundation for all future collaborative development, ensuring that every modification is traceable and that the development environment is ready for complex team-based workflows.