Git and GitHub:

1. What is Version Control?

Imagine you are writing a 50-page report. You save it as `report.doc`. Then you make changes and save it as `report_v2.doc`, then `report_final.doc`, and finally `report_REAL_final_v3.doc`.

Version Control is a system that does this automatically and professionally. It tracks every single change made to a file so you can:

- See exactly what changed and who did it.
- Go back to an older version if you make a mistake.
- Work on the same project with 100 other people without messy file naming.

What is Git?

Git is a **Version Control System (VCS)**. It is a tool that runs locally on your computer to track every single character you change in your project files. It acts like a "Save" button that remembers every version of your file since the day you created it.

What is GitHub?

GitHub is a **Web-based Hosting Service**. It is a giant server in the cloud where you store your Git repositories. While Git tracks the changes, GitHub provides a social interface, a backup location, and tools for teams to work together.

Why do we use them together?

- **Git** handles the "How": It manages the technical history of your files.
- **GitHub** handles the "Where": It gives that history a home where other people can see it and contribute.

Git vs. GitHub: The Difference

Feature	**Git**	**GitHub**
What is it?	A software (tool) installed on your computer.	A website (cloud platform) that hosts your code.
Where is it?	It lives **Locally** on your hard drive.	It lives **Online** in the cloud.
Purpose	To track changes in your files.	To share your files and collaborate with others.
Internet ?	You don't need the internet to use Git.	You **must** have the internet to use GitHub.

Git Concepts

- **Repository**- A directory where git monitors your project files and records their revision history.
- **Clone**- It created a local copy of a remote repository on your machine.
- **Stage** - It Selects specific changes that you want Git to include in the next snapshot.
- **Commit-** It records the staged changes as a permanent version in the project history.
- **Branch**- Lets you develop new features or experiment without affecting the main project.

- **Merge-** Integrates changes from one branch into another.
- **Pull-** It fetches and applies updates from a remote repository to your local one.
- **Push-** It Uploads your local commits to a remote repository
-

Git Repository Structure
It consists of 4 parts:

1. **Working directory:** This is your local directory where you make the project (write code) and make changes to it.
2. **Staging Area (or index):** this is an area where you first need to put your project before committing. This is used for code review by other team members.
3. **Local Repository:** this is your local repository where you commit changes to the project before pushing them to the central repository on GitHub. This is what is provided by the distributed version control system. This corresponds to the .git folder in our directory.
4. **Central Repository:** This is the main project on the central server, a copy of which is with every team member as a local repository.

4. Essential Git Commands

Setting Up

Before you start, you tell Git who you are (so your name appears on your work):

- git config --global user.name "Your Name"
- git config --global user.email "you@example.com"

Starting a Project

- **git init**: Turns a regular folder into a Git Repository.
- **git clone <url>**: Copies an existing project from GitHub to your computer.

Saving Your Work (The Daily Workflow)

1. **git status**: Tells you which files have changed. (Check your status often!)
2. **git add <filename>**: Moves a file to the **Staging Area**.
    - *Pro-tip:* Use git add . to stage all changed files at once.
3. **git commit -m "Your message here"**: Saves a snapshot of your staged files with a note explaining the change.

Syncing with GitHub

- **git push origin main**: Sends your local saves up to the cloud (GitHub).
- **git pull**: Fetches the latest work from GitHub down to your computer.

5. Key Concepts: Branches and Merging

A **Branch** is like a parallel universe. You can create a branch to try a new idea without breaking the "Main" (working) version of your code.

- **git branch <name>**: Creates a new branch.
- **git checkout <name>**: Switches you to that branch.
- **git merge <name>**: Combines the changes from your side-branch back into the main project.