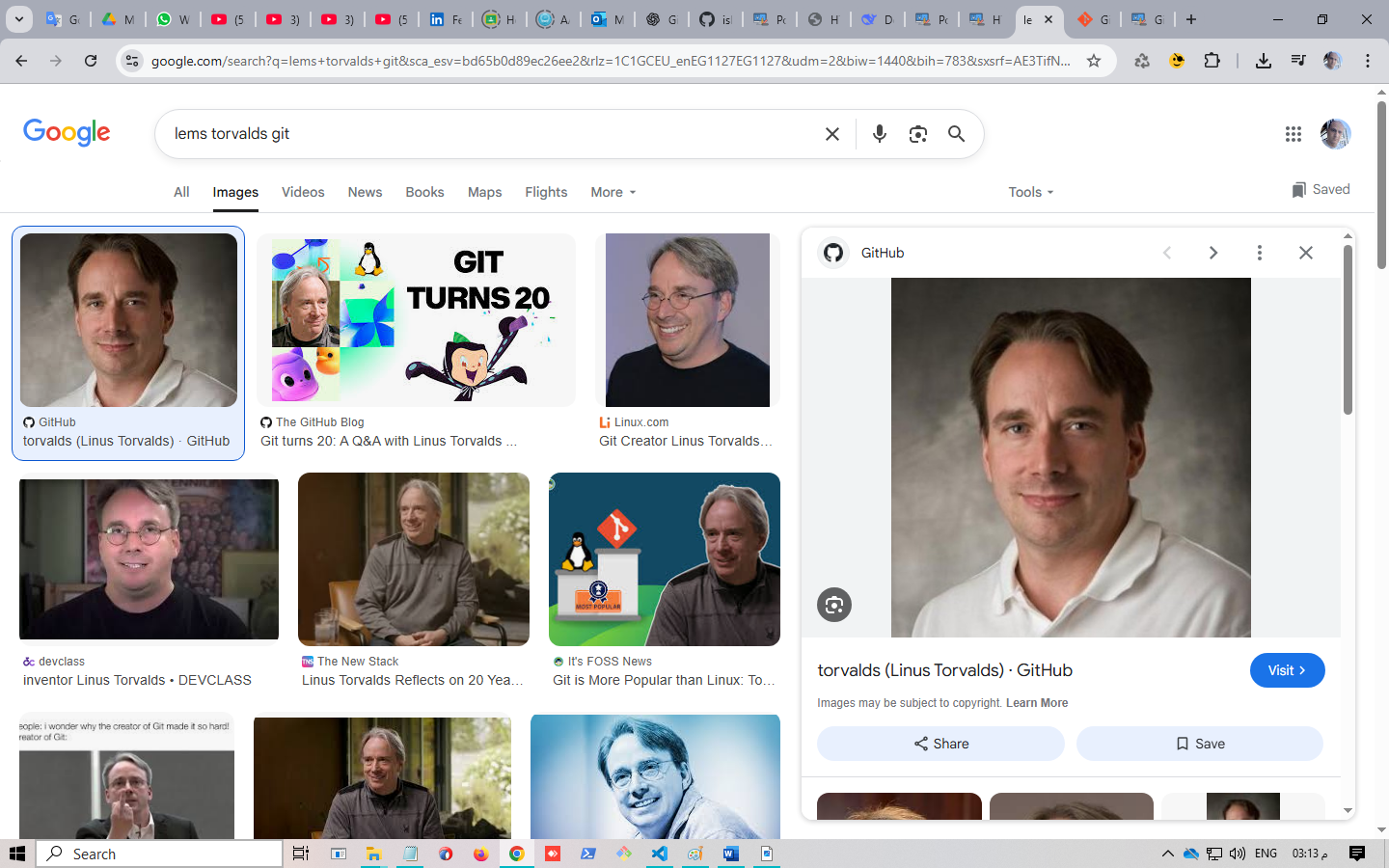
**Introduction of version control**

****

**Git** is a tool used for **tracking changes in your project code**. It helps developers manage versions of their code, collaborate with others, and keep a history of all modifications made over time.

**Who created Git?**

* **Linus Torvalds**, the creator of Linux.

**When?**

* **April 2005**

**Why was Git created?**

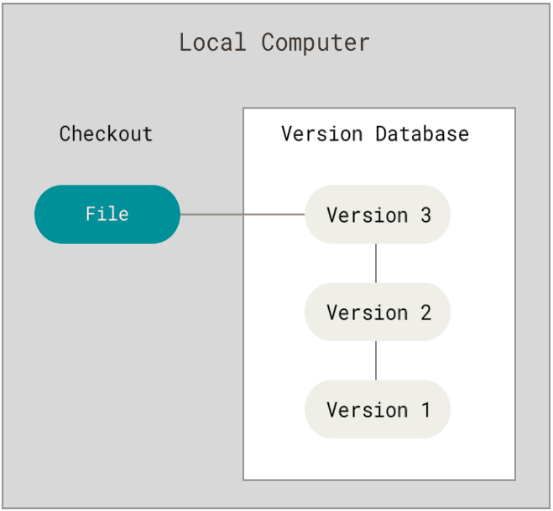
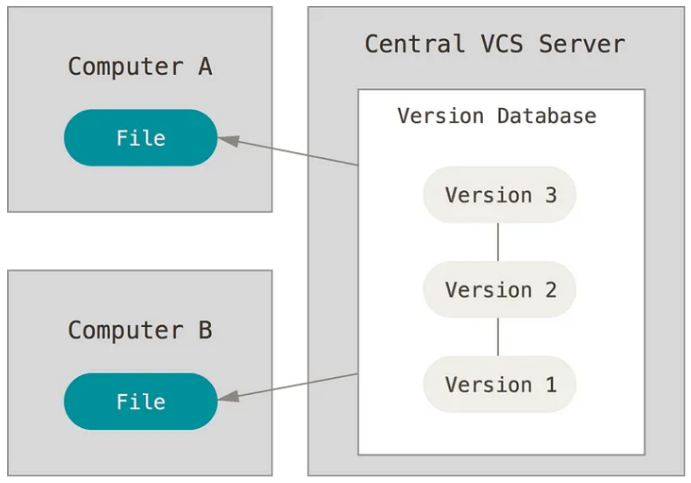
Before Git, the **Linux kernel team** used a proprietary version control system called **BitKeeper**. But:

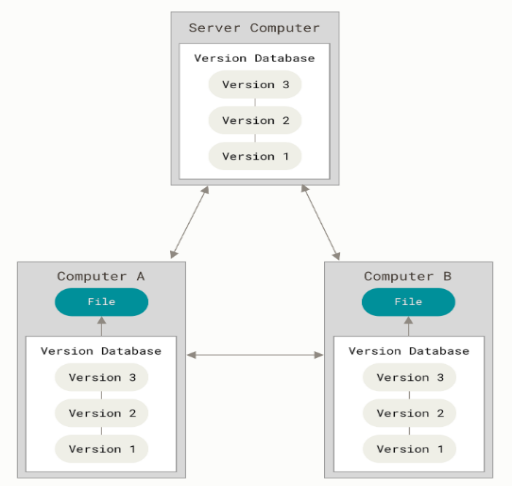
* BitKeeper was **closed source**.
* In 2005, the free license for Linux developers was **revoked**.
* Linus needed a **fast**, **distributed**, and **open-source** tool to manage changes in the Linux source code.

So, he built **Git**.

There are three versions Control from git

* Local Version Control Systems (LVC)
* Central Version Control Systems (CVC)
* Distribution Version Control Systems (DVC)





Git Basics

* Different than others
* Snapshots not differences

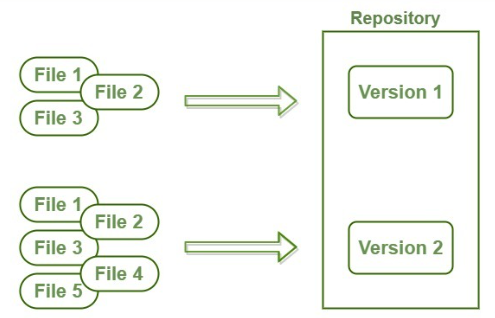
How Git Actually Tracks Changes:

Git doesn't store just the changes (like some other version control systems — e.g., SVN or traditional diffs). Instead:

* Every commit is a snapshot of your entire project at that moment.
* BUT — Git is very efficient. It only stores new data for changing files.
* Unchanged files in the snapshot are just pointers to their previous versions.

Requirments

File a.txt is a version 1 in folder **working directory** (working tree) or repository (repo) After sometime and after making some altert and the addeding in a.text file we will arriv to version 2



**Note:** In Git, you're **working directory** shows only **one version** of the file (usually the latest one you're working on). However, the **Git repository** stores **all versions** (history of changes) of the file.

* Track everything (content and metadata)

Git tracks not only the content of files but also metadata such as timestamps, authors, commit messages, etc.

* OS independent

**Git works across all major operating systems — Windows, Linux, and macOS —**  
because it stores everything as **folders and files** using **lightweight encryption and compression**, which makes it **portable and efficient**

* Unique ID

Every commit is identified by a **SHA-1 hash** (a unique 40-character string), ensuring **integrity** and **non-conflicting versions**.

* Track History (log)

Git maintains a complete **history of changes**, viewable with commands like git log, allowing users to **revert**, **compare**, and **track authorship**.

* No Content change

Git Objects (which Git tracking)

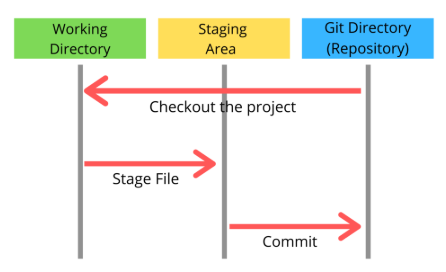
* Blob -> (tracking file & metadata)
* Tree -> (tracking folder & metadata)
* Commit
* Tagging

Note: The **Git repository (. git folder)** is located **inside the working directory**.  
It contains all the **metadata**, **version history**, and **configuration** for your project.

. git -> Hedin file

Git uses three**-tree architecture** to manage and track changes. These three "trees" represent different areas where files exist during the Git workflow:

1. **Working Directory (Working Tree)**
   1. This is **your local workspace** — the files and folders you see and edit.
   2. It reflects the **current checked-out version** of the project.
2. **Staging Area (Index Tree)**
   1. Also called the **Index**, it's where you place changes you want to commit.
   2. Acts like a buffer zone.
   3. You manually add files here before committing.
3. **Git Directory (HEAD Tree / Repository)**
   1. This is the **actual Git database** where committed snapshots are stored.
   2. HEAD points to the latest commit in the current branch.



Cases of file or folders in git

* 1. **Untracked**: Not in staging or commits
  2. **Modified (but not staged)**: Changed but not staged
  3. **Staged (Ready to be committed)**: Added to staging, ready to commit
  4. **Committed (Tracked and Saved)**: Saved in repository (HEAD)
  5. **Deleted**: Removed from working directory
  6. **Renamed or Moved**: Moved or renamed file

**Command => git init**

The git init command is used to **create a new Git repository**. It **initializes** a .git directory in your project, making it ready to track changes using Git.

In this step still folders and file in working directory untracked

git init -> Create repo

**Command => git add**

The git add command is used to **move changes from the Working Directory to the Staging Area**. It tells Git which changes you want to **include in the next commit**.

in this step make folders and files tracked

git add <file> # Add one file

git add. # Add all files in current directory

git add folder/ # Add all files in a folder

git add \*.html # Add all HTML files

git add -A # Add all changes (new, modified, deleted)

git add -p # Add part of file interactively (patch mode)

Note: In the staging step, files and folders are *indexed*

**What does "indexed" mean in Git?**

* When you run git add, Git **adds metadata and content info** about the files to a special structure called the **Index**.
* The **Index** is an internal database that tracks what will be committed next.

After you’ve already initialized Git and made an initial commit, any **new change** to a tracked file makes it go into the **Modified** state.

**Command => git commit**

The git commit command is used to **save (record) changes** from the **Staging Area (Index)** to the **Git repository (HEAD)**. It's like taking a snapshot of your project at that point in time.

What Is **Git** and **GitHub**?

* Git is Distributed version control system
* Git is Free and Open Source
* GitHub is Source for Project and Sources [GitLab, Bitbucket]
* GitHub Simplify Using Git
* You Can Use Without GitHub
* Git Has GUI

Why You Must Learn Git?

* Devs Contribute to the same Project
* You Can revert changes
* You can collaborate to create new features
* You can solve conflicts
* You can organize features

Words you will hear

* Repository (repo)
* Branch
* Local Repo
* Remote Repo
* Commit (snapshot or checkpoint in your local repo)
* Clone [ from local or remote]
* Push [Upload local changes to remote]
* Pull [You pull changes from remote repo to your local]
* Pull Request [Tell other About your changes to pull it from local to Remoremote]