Introduction to Git

1. Version Control

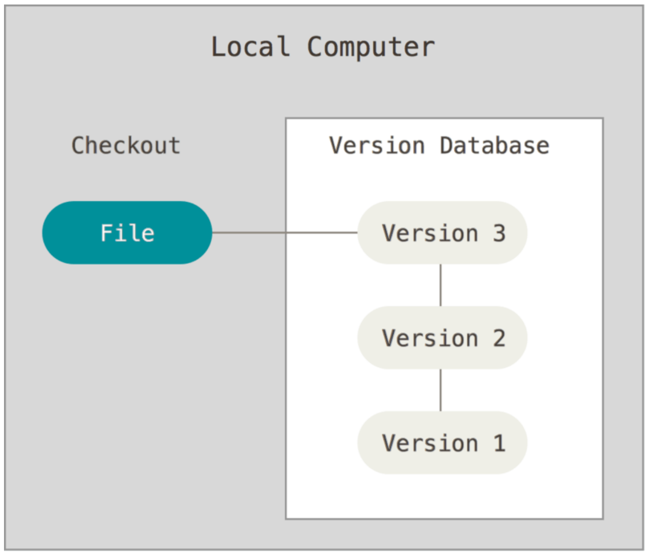
Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. A Version Control System (VCS) allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, etc.

E.g.: Git, CVS, Subversion, Mercurial, Perforce etc.

2. Evolution of Version Control Systems

a) Local Version Control Systems:

It was a simple local database, which kept all the changes to files under revision control.



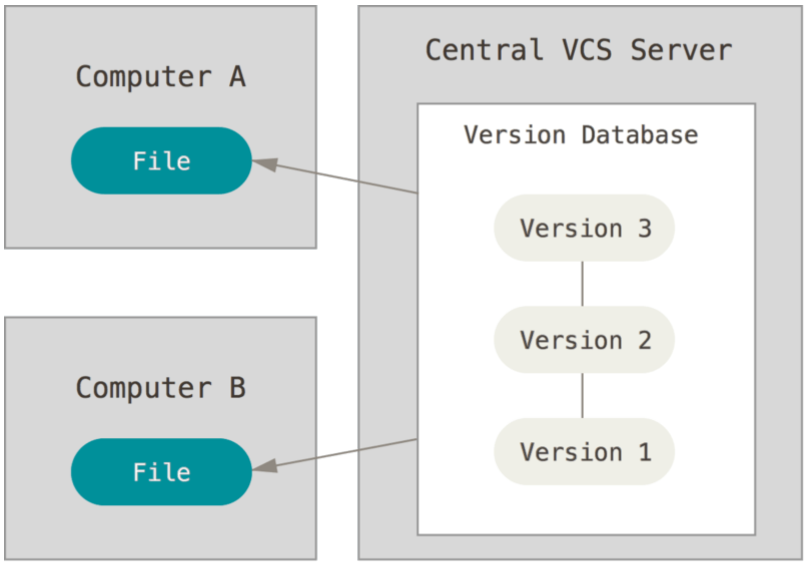
b) Centralized Version Control Systems:

It was a single remote server that contained all the versioned files, and a number of clients that checked out files from that central place.

E.g.: CVS, Subversion etc.

Advantage: Helped in collaboration between developers.

Disadvantage: Single point of failure.



c) Distributed Version Control Systems:

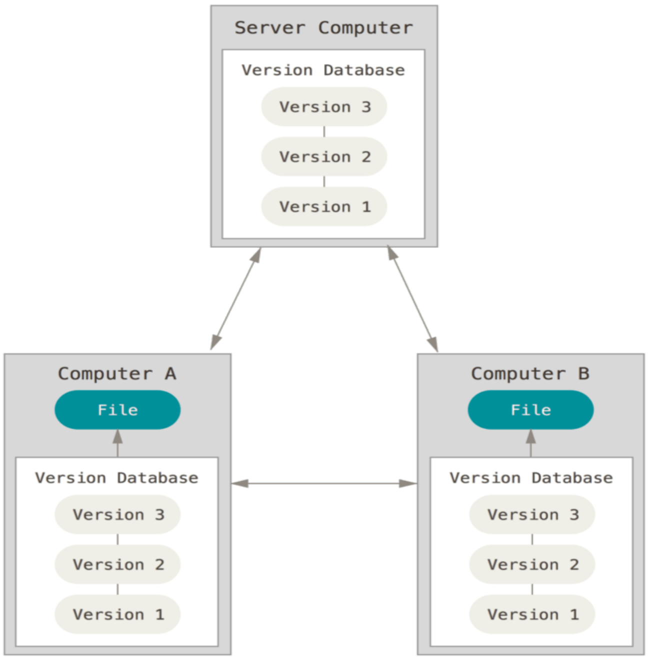
Clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Every clone is really a full backup of all the data.

E.g.: Git, Mercurial etc.

Advantages:

Helped in collaboration between developers.

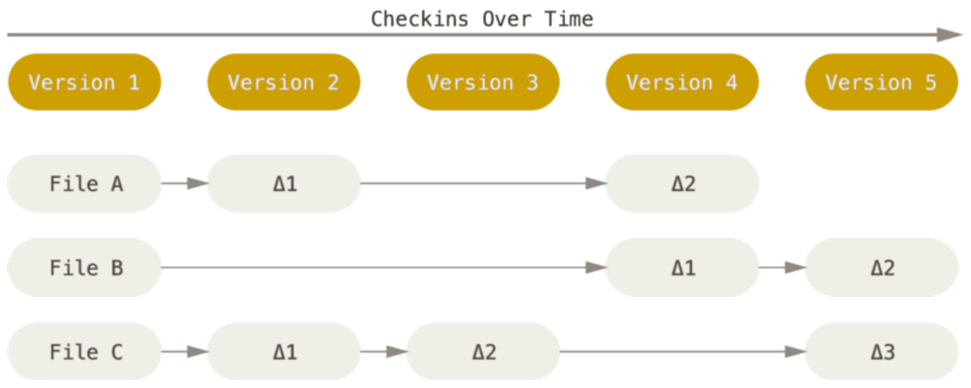
No single point of failure.



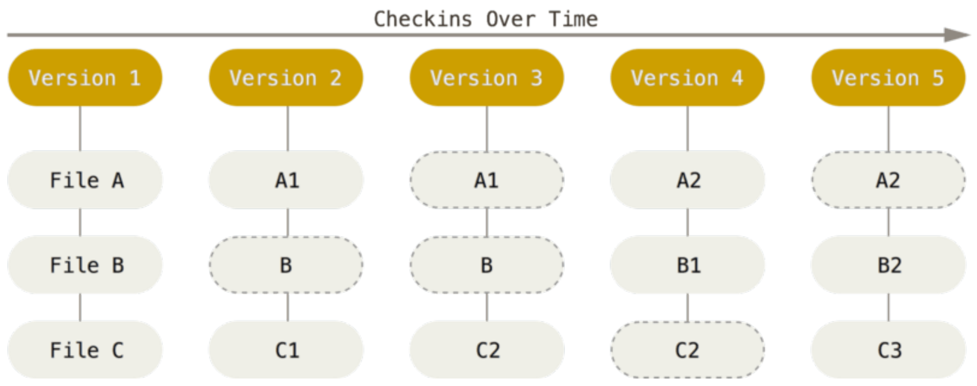
3. What is Git?

Git is a free and open source distributed version control system.

Snapshots, Not Differences: The major difference between Git and any other version control systems is the way Git thinks about its data. Most other systems store information as a list of file-based changes i.e. they think of the information they store as a set of files and the changes made to each file over time (this is commonly described as delta-based version control).



Git thinks of its data more like a series of snapshots. With Git, every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored. Git thinks about its data more like a stream of snapshots.



Everything in Git is checksummed: The mechanism that Git uses for this checksumming is called a SHA-1 hash. This is a 40-character string composed of hexadecimal characters (0–9 and a–f) and calculated based on the contents of a file or directory structure in Git. Hence it's impossible to change the contents of any file or directory without Git knowing about it.

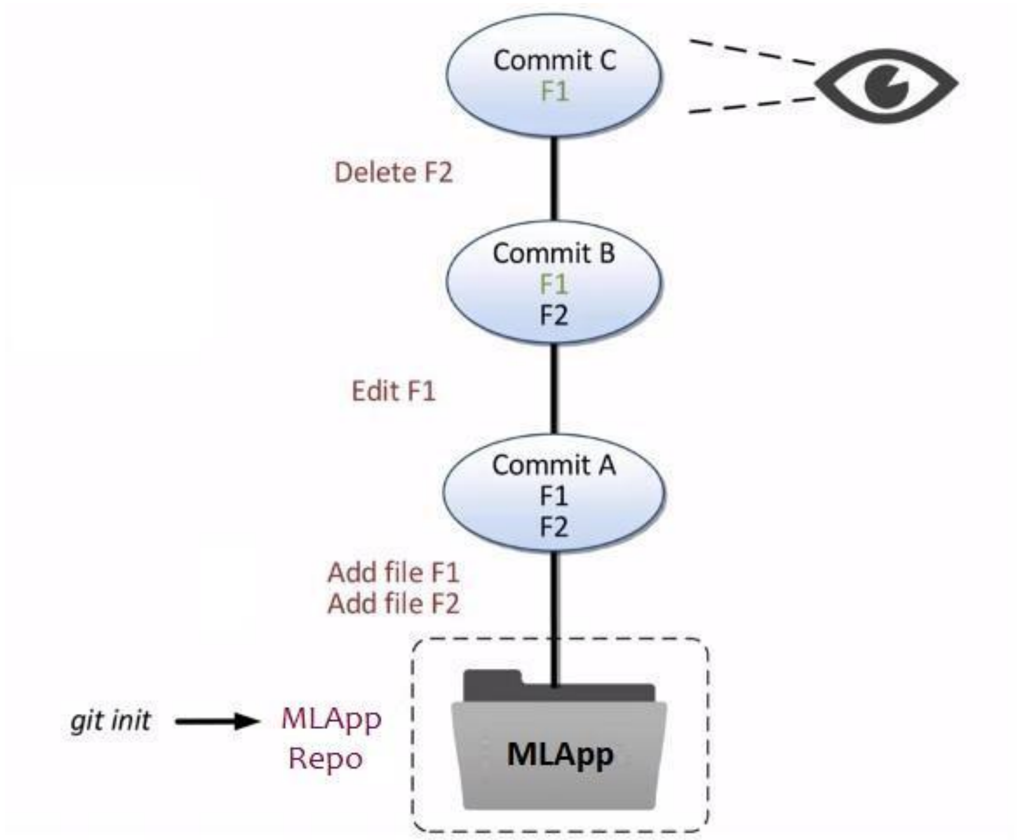
Git Commit Graph: Git keeps track of the changes in files over time. It does this by enabling us to take snapshots of the files any time. These snapshots are called commits. We can represent these commits by a basic graph called the commit graph.

Let's imagine we are setting up a new Git repository. We need to first create a standard directory on our file system. We can convert this directory into Git repository by running the ‘git init’ command. Then let's add two files (F1 and F2) into the working directory. After adding the files, we can take the first snapshot (Commit A).

Now let’s make some changes in the file F1. After editing F1 we can capture the new changes in F1 with the second commit (Commit B). So now we have two commits in our commit history.

Let’s say if we want to delete the file F2, let’s make a third commit after removing the file F2 (Commit C). Now we have built a history of three commits. The third commit contains only the file F1, since we have removed it after the second commit. In our working directory we will only see the file F1.

Since we have all the snapshots, we can restore earlier versions of the files as and when required. For example if we want F2 back, we can restore it from the second commit (Commit B). Every commit is logged by Git. It will log who made the changes and when was the changes made.

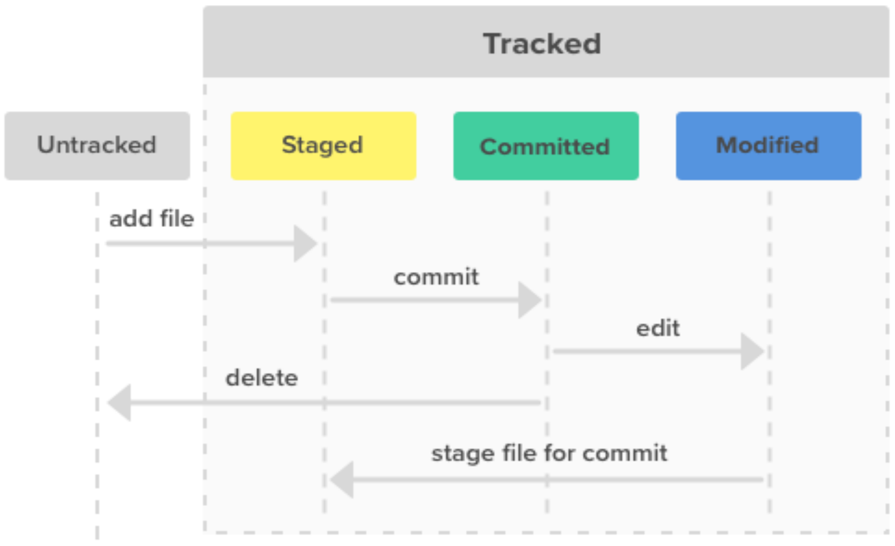


In Git each file can be in one of two states:

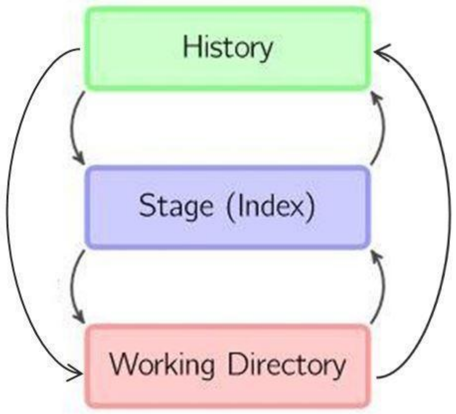
- Tracked
    - Tracked files are those files that were in the last snapshot; they can be:
        - **Modified** means that you have changed the file but have not committed it to your database yet.

        - **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.
        - **Committed** means that the data is safely stored in your local database.

- Untracked
    - Untracked files are files in your working directory that were not in your last snapshot and are not in your staging area.



Git has three main sections



- **The Working Tree (Directory)** is a single checkout of one version of the project. These files are pulled out of the compressed database in the .git directory and placed on disk for you to use or modify.
- **The Staging Area (Index)** is a file, generally contained in your .git directory, which stores information about what will go into your next commit.
- **The Git History** is in the .git directory and this is where Git stores the metadata and object database for your project.

4. The basic Git workflow

1. Modify files in your working tree.
2. Selectively stage just those changes you want to be part of your next commit, which adds only those changes to the staging area.
3. Do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your git history.

References:

1. https://git-scm.com/book/en/v2
2. https://www.youtube.com/watch?v=uR6G2v_WsRA