Here’s an improved version of the text with better grammar, clarity, and flow:

Version Control Tools

Introduction to VCS

1. Version Control Systems

Version Control Systems (VCS) are tools that track changes made to files or sets of files over time, allowing easy retrieval of earlier versions. They can be used with any type of file, not just source code.

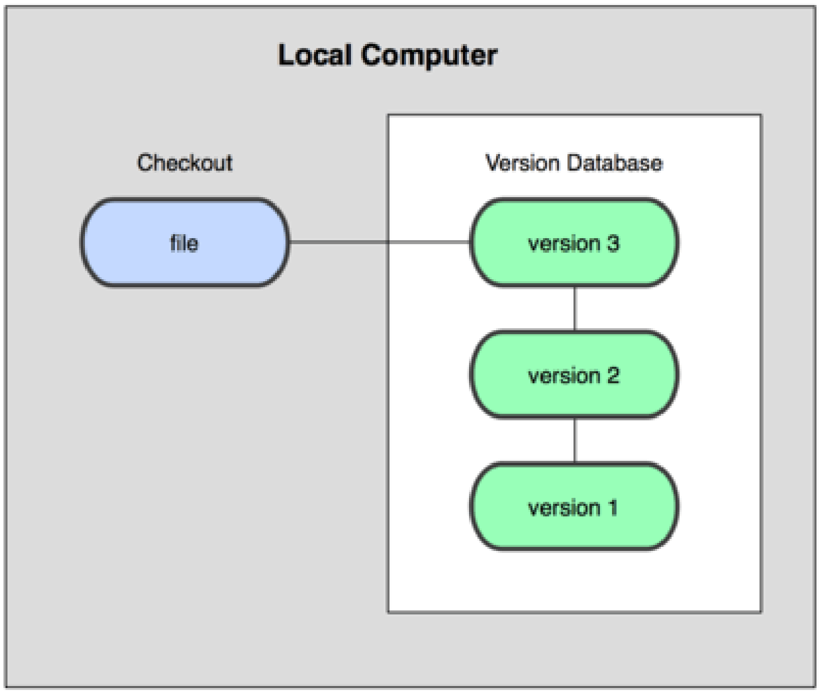
A VCS enables users to revert to a previous state of a file or an entire project, compare changes over time, identify who made changes and when, and more. Additionally, if a file becomes corrupted or is accidentally deleted, it can be restored to a previous version.

1.1. VCS Types: Local VCS

VCS can operate either locally or online. Local VCS is useful for creating backups of a project stored on the same machine. This allows for quick restoration to a stable version if needed, such as in case of an error.

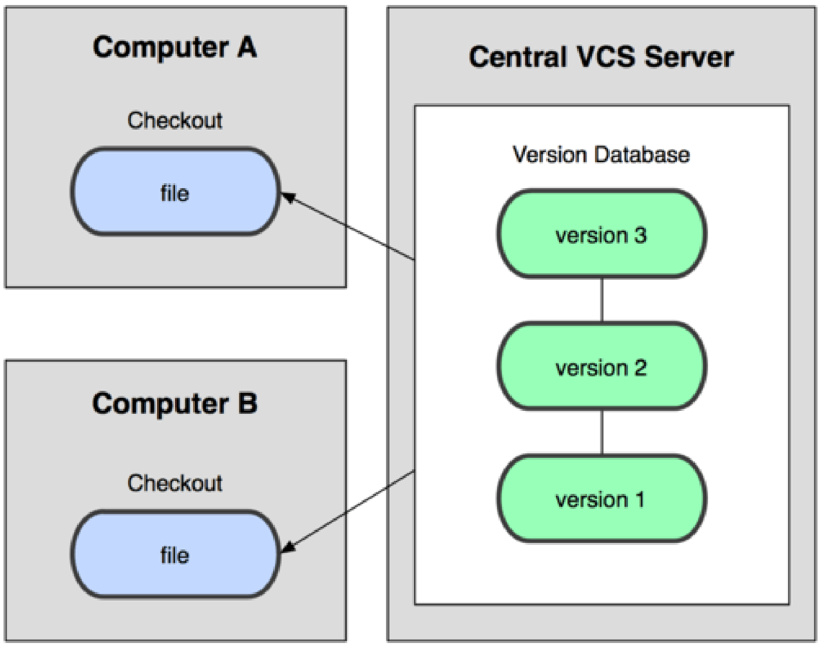
The main advantage of local VCS is simplicity, while the main drawback is that version control must be managed manually, increasing the risk of mistakes. For example, you could mistakenly modify a backup file instead of the current version.

To address these issues, some tools help manage changes more effectively. One of the most well-known is *rcs*, a system still available on many computers. It stores a series of patches (differences) between versions of files. These patches are saved in a special file format, allowing the system to reconstruct previous versions by applying or reversing the patches.



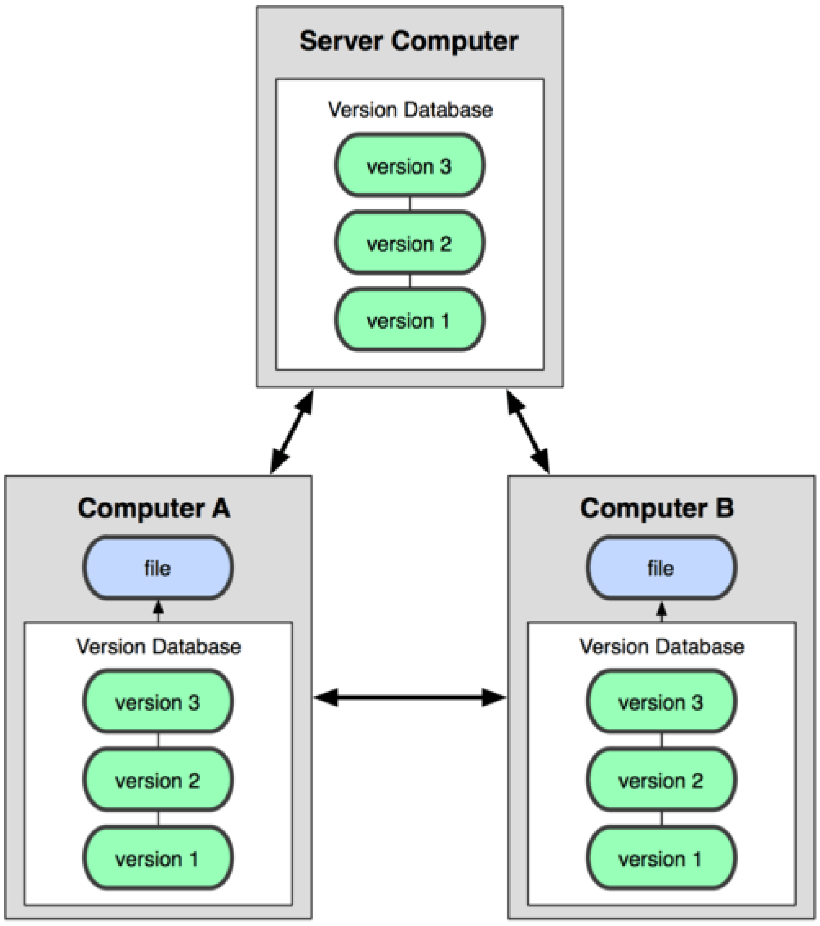
1.2. VCS Types: Centralized VCS

Local VCS is not ideal for collaboration among multiple team members. Centralized VCS (CVCS) solves this problem by using a single server that stores all files and their version history. Multiple clients can connect to the server to upload and download changes. This approach became standard due to its advantages over local VCS, but it has a significant drawback: if the server fails, the entire project may be lost.



1.3. VCS Types: Distributed VCS

Distributed VCS (DVCS) emerged to overcome the main limitation of CVCS. In a DVCS, such as Git, Mercurial, Bazaar, or Darcs, clients download the entire repository, not just individual files. This means that if the server fails, any local copy of the repository can be used to restore the project. Essentially, each download acts as a complete backup.



2. Git

Git was created by the Linux development team after discontinuing the use of *BitKeeper*, their previous version control tool. Based on BitKeeper's limitations, they defined several key objectives for the new system:

- Speed
- Simplicity in design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed nature
- Scalability for large projects (like the Linux kernel)
- Efficiency in speed and data size

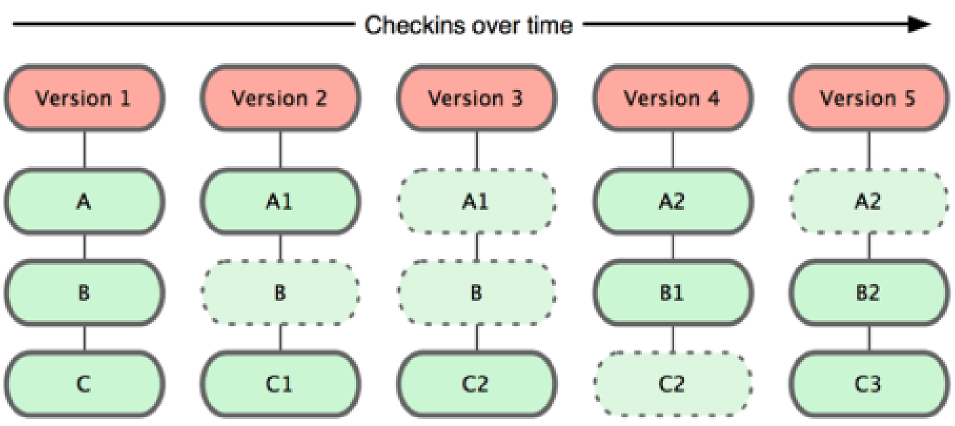
Since its launch in 2005, Git has evolved significantly, becoming easier to use, faster, and more efficient with large projects, and featuring an outstanding branching system.

2.1. Git Fundamentals

Let’s explore some of the basic concepts of the Git version control system.

Data Modeling

Git stores a series of snapshots of the file system rather than just a list of changes. Whenever a new change is uploaded, Git takes a "snapshot" of the current state of every file and saves a reference to this snapshot. If a file hasn’t changed, Git doesn't create a new copy but instead points to the previous version.



This approach sets Git apart from most other VCS, making it more like a small file system with versioning tools than a traditional VCS.

Local Operations

Most Git operations are performed using local files and resources. Since the project history is stored locally, many actions are instantaneous, allowing you to work on a project even without an internet connection. Changes are saved locally, and the remote repository can be updated once the connection is restored.

Integrity

Git uses the SHA-1 hashing algorithm to store information, ensuring data integrity. If any data is altered, Git will detect the change.

Additive Only

Git operations are additive, meaning that information is never erased, only added. This makes it easy to undo changes, as previous data is preserved.

Project States

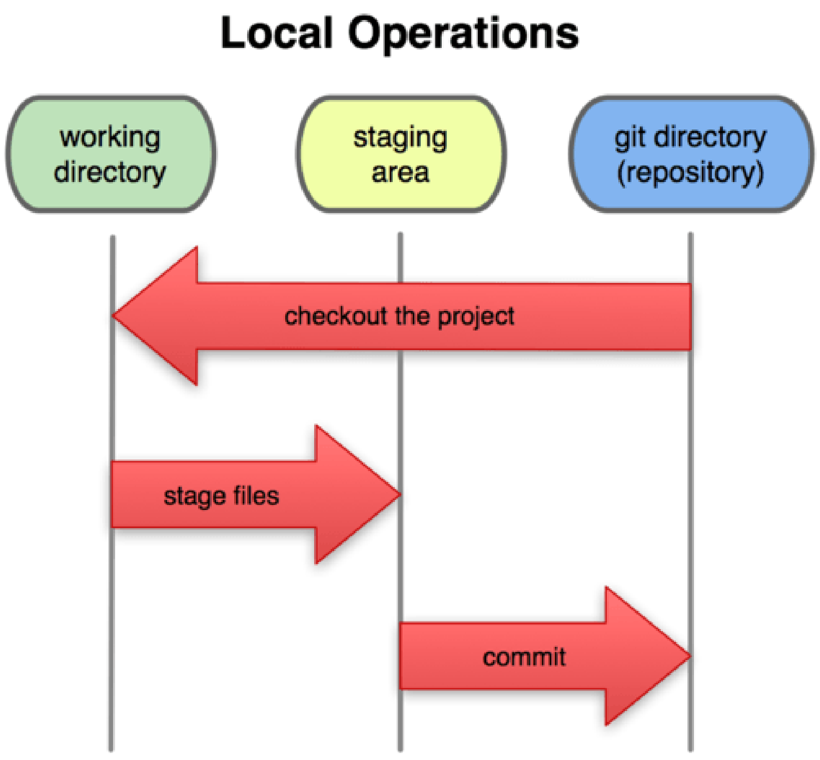
Files in a Git-managed project can exist in one of three main states:

- **Modified**: The file has been changed locally but hasn't been committed.
- **Staged**: The file is marked for inclusion in the next commit.
- **Committed**: The file is safely stored in the local Git repository.

Accordingly, Git projects have three main sections:

- **Git Directory**: Where Git stores the metadata and the project database. This is what gets copied when cloning a repository.

- **Working Directory**: A local copy of a project version, extracted from the Git database and placed in a folder.
- **Staging Area**: A file in the Git directory that stores information about what will be included in the next commit. It's also known as the *index*.


Local Operations
working directory
staging area
git directory (repository)
checkout the project
stage files
commit