ENSE 375 Version Control Report
Git VCS

Li Pan

Git is an open-source version control system that is robust and reliable. It is widely used today for tracking and managing changes to source code during the software development process.

Git History

The Git development environment was first created in the Linux kernel. In the early 2000s, distributed version control systems, such as BitKeeper and Monotone, managed the codebase in the Linux kernel for free. However, the creators of both systems severed ties with the Linux kernel, and new scripts to manage email patches needed to be developed. As these scripts were created, they needed to be quickly merge-able such that developers could continue modifying the code base. This intended functionality was the birth of the Git version control framework.

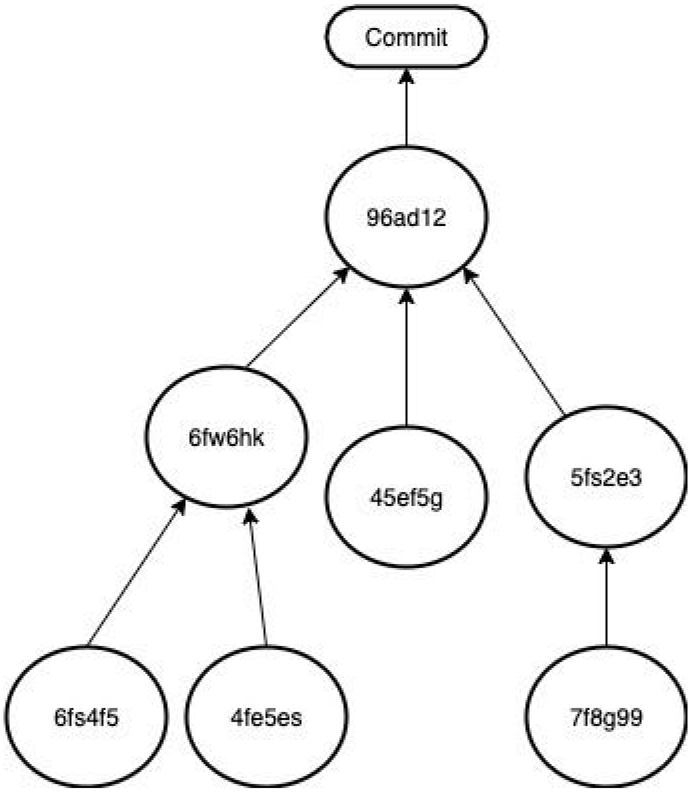
The initial design goals of Git were heavily inspired by BitKeeper; support for distribution of versions, security, and tracking workflow history were considered paramount. Moreover, the architecture of Git was structured such that collaborators could work in individual branches while the overarching repository would log all changes and arrange merges between branches. Git's primary competitor during its early phases of development was Mercurial, an open-source version control system that is used by Facebook. In the modern era, the effectiveness of Git has influenced the creation of additional open-source version control software, such as Fossil and Bazaar. However, Git remains the most popular and widely-used service in its domain.

Version Control

Version control is extremely useful for the software development process. It enables software developers to work on the same project simultaneously, tracks the exact changes of each modification, and supports backups of older versions of the source code. Most version control systems can restore and track changes to content, but only a select few are capable of analyzing the distribution of labour between collaborators. Git contains all major features of version control systems; it supports content storage, commit records, merge histories, and management of different distributions.

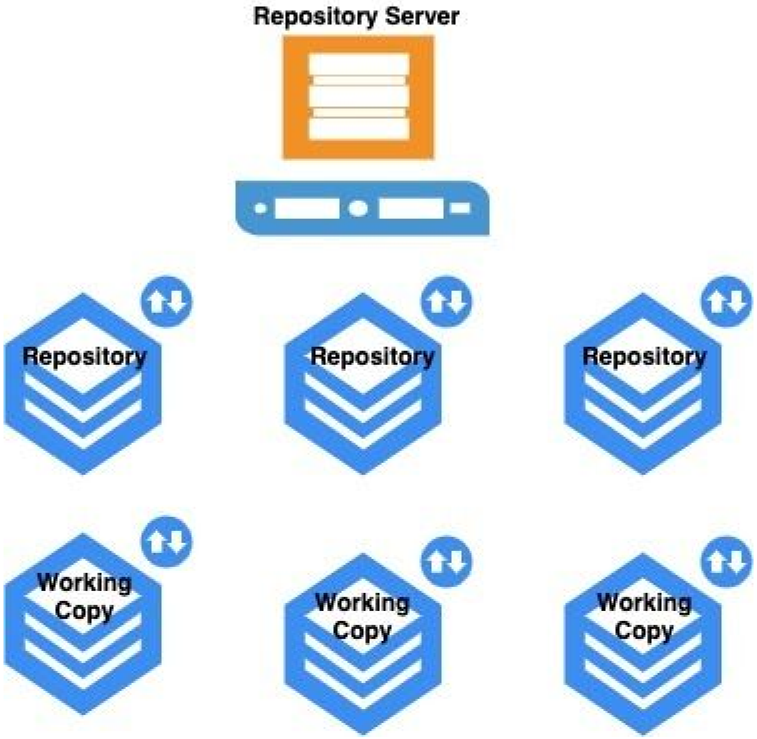
Git uses directed acyclic graph (DAG) design to store content with different types of objects. In comparison to linear history, the DAG is advantaged with respect to content restoration. It can capture differences between each version, store these differences as snapshots, and restore any previous version of a branch in the repository’s history.

Commits and merge history can be approached linearly or with the DAG. Linear history only allows sequential commits on unmerged branches, while the DAG remains cognizant of the change history, which enables unlimited use of commits. Each commit contains metadata, and each commit can contain many parents. Git also records a repository’s merge history.



All commit nodes are directed to the root directory. Each commit can have multiple file changes, and the commit node can have multiple parent branches. When a directory node has different branches, these branches are guaranteed to include the same content. When different branches are merged together, Git can compare the differences between content and keep changes together.

Generally, version control systems can be localized, centralized, or distributed. Localized systems are simple in structure and are only used on a local computer. It is difficult to track which directly is currently being worked on; thus, the localized version control system can easily cause errors. Centralized version control systems enable multiple collaborators on the same project. The server contains one repository with all previous versions of the project. However, if a developer commits a change that corrupts this server, then all other developers are denied access to the project. In addition, the changes made by each developer on their local computer will be lost. Distributed version control systems overcome both of these drawbacks. In distributed version control, every developer possesses a complete copy of the repository in their branch, which contains all metadata. Developers only perform modifications in their own branch, ensuring that any errors that they produce will only affect their branch, rather than the entire repository. Git is a distributed version control system (see Figure 2), and it contains advantages which provide backups of each collaborator's branch, as well as straightforward integration of each branch with the newest version of the project on the repository's server.



Toolkit

Git was originally designed for Linux, and its toolkit includes many powerful terminal commands. The low-level commands track and manage content changes with directed acyclic graphs, while the high-level commands manipulate the different repositories. Many graphical user interface (GUI) tools are built overtop of this primary toolkit.

Repository, Index, and Working Area

A new repository can be created by running the *git init* command. It is easy to create a branch, commit, and tag with the repository, and Git allows different local repositories to communicate with one another. Under a Git directory, there are many different categories. For one, Git configuration is the description of the local repository. It can be checked under git/config, git/description, and git/info/exclude. Next, Git's hooks directory contains the scripts which run based on certain events in the repository. Third, Git staging can be found in the git/index path. Finally, Git's object database is found in the git/objects directory, which contains all content and pointers to local content.

The Git index file is used as a staging area between the working copy and the local repository, and lists the changes made during the staging. The *git status* command can check the current index and indicate all files that are modified on staging. In the Git working area, the developer can modify files, and these modifications can be kept in the staging area with the *git add* command. The object ID is recorded as well.

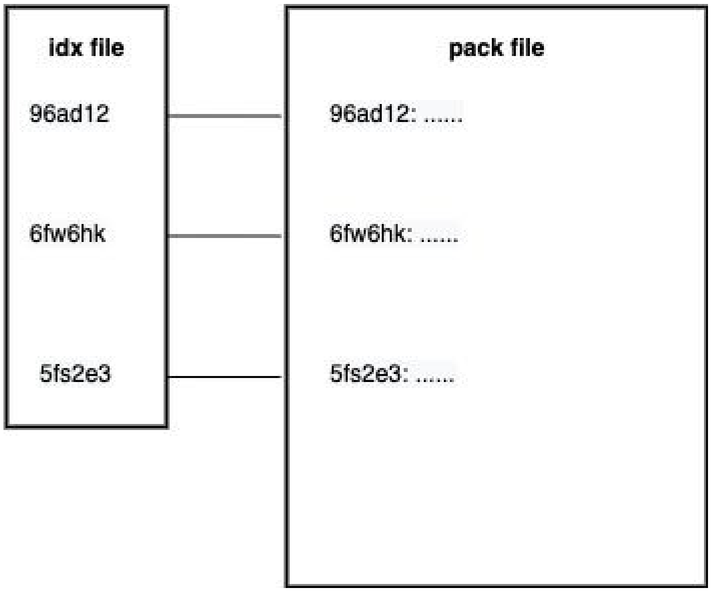
The Object Database

The object database is a set of directories in the Git system which allows developers to retrieve content. Git contains four basic objects: tree, blob, commit, and tag; each of which has a type, size, and content. A tree is a representation of a directory. A blob is a representation of a file in a directory. A commit contains a snapshot's information, and points to its top-level tree. Finally, a tag acts as a label on a commit, which contains the name in the repository history.

All objects are referenced via a simple hashing algorithm (SHA) in Git. If two objects are identical, then they possess the same hash value. Two properties of the SHA point to the same object representation in Git's distributed model.

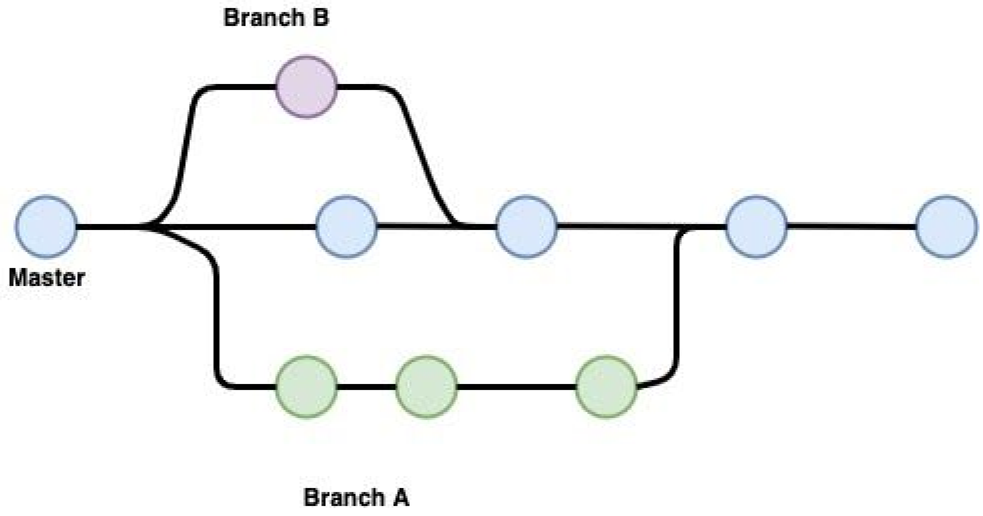
Storage and Compression Techniques

Git manages storage space by packing objects in a compressed format. Each index file corresponds to a pack file (see Figure 3). Git efficiently manages data compression at the blob object level; each commit only takes up a small amount of space. Git can calculate the commits and content that need to be sent, and then generate a pack file by using a client-desired protocol for communication.



Merge History

Git can merge completed changes from a working branch into the master branch. If this occurs, then the development history will display two nodes joining together into a new version. Each developer can merge their contributions separately without the risk of affecting others' work.



Git also supports a visualized merge history, which can be viewed through the *git log -graph* command.

Conclusion

Overall, Git supports many popular integrated development environments, and performs as an extremely powerful version control system. Numerous programming platforms have become integrated with Git, including NetBeans and Visual Studio Code. Ultimately, Git drastically assists software developers in improving their work efficiency.