Robert Adams

Git is an open source distributed version control system created in 2005 by Linus Torvalds. While developing the Linux kernel, frustrations caused by other version control systems(VCS) led to the creation of Git. Git was meant to differ from many of the existing VCSs by supporting distributed workflows, offering safeguards against data corruption, and offering high performance. Additionally, Git, like other distributed VCSs, allows collaborators to work offline and commit their work incrementally, allows collaborators to determine when work is ready to share, offers the collaborator access the history of the repository when offline, and allows work to be published to multiple repositories. GIt, as with most VCSs, was developed mainly for software development. Software developers use Git for collaborative projects, allowing them to work individual and merge and commit their code to the repository.

Git stores data and commit/merge history with directed acyclic graphs. As opposed to linear history, each Git commit can have zero or multiple parent commits. The content stored in the commits are also directed acyclic graphs where the directory structure is represented. When merging, Git merges the two directed acyclic graphs, and Git is much more efficient than other VCSs at finding common ancestor during these merges. Data is also compressed into packs which solve the problem of inefficient use of space.

Git follows the distributed model for VCSs, meaning collaborators can make commits on local repositories and later push those commits to a pubic repository. This allows offline work which can later be shared on the public repository. Changes from local repositories can be merged into the public repository. Developers are given a full history of each commit and merge.

Having a Toolkit design, Git allows for easy compatibility with many tools which are simply built on top of the Git core toolkit. However, developers have complained about the lack of a linkable library, and GUIs and web Interfaces often make calls to Git binary which can be slow. This design has often made it challenging for IDEs to be integrated with Git, while this is not the case for some other VCSs.

There are four main objects in Git: trees, blobs, tag, and commits. A tree is an object which can contain either other trees or blobs. A blob is an object which represents an individual file in a directory. A tag points to another object and can hold any additional data relevant to the referenced object. A commit is an object which points to a tree, contains a time stamp, a log message, and the parent commit objects. Which adhere to low coupling as well as the single responsibility principle as the have specific functions and contain the needed information within the object. Git uses SHA hashes to determine whether objects are identical or not, or when copying objects to determine if there was any data loss or corruption.

As mentioned before, Git uses directed acyclic graphs to store the repositories revision history. Unlike linear time history VCSs, this allows branches to be aware of past merges, letting users see a complete merge and commit history. However, with this approach it is difficult to determine which commits were contained in which branch.

Git has a mild usability problem. The design for Git is relatively simple compared to some other VCSs, but, unlike those, Git expects you to understand how it works for you to be able to use its full functionality. This inherently makes it difficult for new people to use Git, as just memorizing a couple commands isn’t quite good enough sometimes. Also, unlike non-distributed VCSs, Git requires you to push your commits to the remote repository. This requires the developer to decide when and remember to share their content publicly.

Git certainly achieves the level functionality envisioned by its creators, as it accomplishes its main goals through non-linear history and content storage, using a hash function (SHA-1) to help quickly detect data corruption, and using directed acyclic graphs as well as SHA (short circuiting nodes with similar content) to offer greater efficiency. However, Git is rather lacking when it comes to usability. New users often get confused as Git’s toolkit design comes with the consequence of lots of commands and subcommands, and often low level error messages can be difficult to understand. For reliability, as mentioned before, Git safeguards against data corruption by checking packs with SHA. Also, since Git is a distributed VCS with local repositories, as well as a server repository, they can serve as backups for one another. To improve performance, Git implements directed acyclic graphs for storage to help with merging, as well as SHA hashes to help short circuit nodes with identical data. In terms of security, the use of SHA-1 while implemented to safeguard against corruption also provide adequate security.

While sometimes sacrificing usability for functionality, reliability, and performance, Git is still perhaps the best VCS out there. The design allows for non-linear file history, safeguarding against data corruption while hardly sacrificing performance, and data is compressed into packs to dramatically reduce the otherwise inefficient use of space. The toolkit design allows developers to build on the core Git toolkit easily, if needed. However, Git has a couple flaws such as the difficulty of integrating with IDEs, as well as usability, especially for new users. The overall design of Git is excellent in that it accomplishes what it was created for, and is now one of the most used version control systems.

http://www.aosabook.org/en/git.html

https://en.wikipedia.org/wiki/Git