PROOF OF CONCEPT FOR GIT

PREPARED FOR

Vinove Software and Services pvt. ltd

PREPARED BY

Shahrukh Anwar

Web Applications Developer

EXECUTIVE SUMMARY

Git is a distributed version-control system for tracking changes in any set of files, originally designed for coordinating work among programmers cooperating on source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Since 2005, Junio Hamano has been the core maintainer. As with most other distributed version-control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server. Git is free and open-source software distributed under GNU General Public License Version 2.

1. Project Overview

Git's design is a synthesis of Torvalds's experience with Linux in maintaining a large distributed development project, along with his intimate knowledge of file-system performance gained from the same project and the urgent need to produce a working system in short order. These influences led to the following implementation choices.

- Strong support for non-linear development

Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. In Git, a core assumption is that a change will be merged more often than it is written, as it is passed around to various reviewers. In Git, branches are very lightweight: a branch is only a reference to one commit. With its parental commits, the full branch structure can be constructed.

- Distributed development

Like Darcs, BitKeeper, Mercurial, Bazaar, and Monotone, Git gives each developer a local copy of the full development history, and changes are copied from one such repository to another. These changes are imported as added development branches and can be merged in the same way as a locally developed branch.

- Compatibility with existing systems and protocols

Repositories can be published via Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), or a Git protocol over either a plain socket or Secure Shell (ssh). Git also has a CVS server emulation, which enables the use of existing CVS clients and IDE plugins to access Git repositories. Subversion repositories can be used directly with git-svn

- Efficient handling of large projects

Torvalds has described Git as being very fast and scalable, and performance tests done by Mozilla showed that it was an order of magnitude faster than some version-control systems; fetching version history from a locally stored repository can be one hundred times faster than fetching it from the remote server.

- Cryptographic authentication of history

The Git history is stored in such a way that the ID of a particular version (a commit in Git terms) depends upon the complete development history leading up to that commit. Once it is published, it is not possible to change the old versions without it being noticed. The structure is similar to a Merkle tree, but with added data at the nodes and leaves.

- Toolkit-based design

Git was designed as a set of programs written in C and several shell scripts that provide wrappers around those programs. Although most of those scripts have since been rewritten in C for speed and portability, the design remains, and it is easy to chain the components together.

- Pluggable merge strategies

As part of its toolkit design, Git has a well-defined model of an incomplete merge, and it has multiple algorithms for completing it, culminating in telling the user that it is unable to complete the merge automatically and that manual editing is needed.

- Garbage accumulates until collected

Aborting operations or backing out changes will leave useless dangling objects in the database. These are generally a small fraction of the continuously growing history of wanted objects. Git will automatically perform garbage collection when enough loose objects have been created in the repository. Garbage collection can be called explicitly using git gc.

- Periodic explicit object packing

Git stores each newly created object as a separate file. Although individually compressed, this takes a great deal of space and is inefficient. This is solved by the use of packs that store a large number of objects delta-compressed among themselves in one file (or network byte stream) called a packfile. Packs are compressed using the heuristic that files with the same name are probably similar, without depending on this for correctness. A corresponding index file is created for each packfile, telling the offset of each object in the packfile. Newly created objects (with newly added history) are still stored as single objects, and periodic repacking is needed to maintain space efficiency. The process of packing the repository can be very computationally costly. By allowing objects to exist in the repository in a loose but quickly generated format, Git allows the costly pack operation to be deferred until later, when time matters less, e.g., the end of a workday. Git does periodic repacking automatically, but manual repacking is also possible with the git gc command. For data integrity, both the packfile and its index have an SHA-1 checksum inside, and the file name of the packfile also contains an SHA-1 checksum. To check the integrity of a repository, run the git fsck command.

2. Obstacles

1). There are no such obstacles in using and implementing Git based logics and concepts. One has to have some knowledge of using Terminal and basic Linux/Unix based commands.

3. Technical Obstacles

1). There are no such obstacles. One should have little basic knowledge of *Unix/Linux* based terminal commands and little Git based commands

4. Software

1). The system must have Git installed on the host system.

5. Milestones and Reporting

Total estimation of man-hours: 6.5

Milestone	Tasks	Reporting	Hrs	Date
1 - Concept				
1.1	Git Overview- (Git Setup, Git Bash)	None	0.5	31/12/20
1.2	Git Features (Add, Clone, Push/Pull Project)	None	0.5	31/12/20
1.3	Git Changes (Add/Commit Changes, Check Status)	None	0.5	31/12/20
1.4	Project History (Project History, Check Versioning, Getting Previous file)	None	0.5	31/12/20
1.5	Managing Branches	None	2	05/01/20
1.6	Comparison of Files and Branches	None	2	05/01/20
1.7	Exploring History (Using)	None	0.5	06/01/20

6. Sources used

1). Wikipedia

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

2). Git SCM

https://git-scm.com/about