Version Control System

Reece Donovan

B.Sc. Computer Science - Final Year Project

Prof. Ken Brown

University College Cork

November 15, 2022

Abstract

Declaration of Originality

Acknowledgements

Contents

Chapter 1

Introduction

Chapter 2

Background

2.1 What is a Version Control System

A version control system saves modifications done by individual software developers which makes the process easier for them since they can track their work over time. It helps share data between nodes where each node can be kept up to date with updated versions so there is no need for any merge conflicts. The advantages provided by VCS include: it aids in collaboration among programmers, improves efficiency when working on larger groups of mixed products; provides an audit trail; easy branching functionality; simplifies team-based development; understanding who has worked on specific pieces or sections during what period in time but also making sure that all changes are tracked properly providing transparency within teams while maintaining optimal performance levels through streamlined management processes, helping avoid errors arising from inconsistency between versions such as mismatched documents or programming problems resulting from conflicting edit operations

2.1.1 What is the purpose of Version Control Systems

The main purpose of VCSs is collaboration...

2.2 Types of VCS

Over the years, two approaches to VCSs have emerged - Centralized and Distributed. Both approaches are in widespread use today. The centralized approach is based on the client-server model where there is a single central repository that stores the history of all files, while the distributed approach

provides each user with a full copy of the repository. The core difference between the two approaches is that the centralized approach requires a single point of failure, while the distributed approach does not. The centralized approach is also more difficult to scale than the distributed approach.

2.2.1 Centralized Version Control Systems (CVCS)

A centralized Version Control System is a system that enables developers to work together on the same project by storing the main copy of files in a central repository. This system keeps track of all files and saves information in the local repository. CVCS are called centralized because there is only one central server or repository. The server maintains a complete record of issues, while clients only maintain a local copy of the shared documents. All developers make their modifications on the repository through checkout but only the last version of the files is retrieved from the server, which means that any modifications made will be automatically shared with other developers. Users can modify in parallel with their local copy of shared documents and sync with the central server to release their contributions and make them visible to other collaborators. Because centralized version control systems rely on one repository that includes the correct version of the project, it must restrict write accesses so that only trusted contributors are allowed to commit modifications. CVCS has some challenges, such as if the central server is inaccessible, then users will not be able to merge their work at all or save the released modifications. It is also if the central repository is corrupted, everything will be lost. Contributors must be the ones who have writing permissions to perform basic tasks, such as reverting modifications to a previous state, creating or merging branches, or releasing modifications with full revision history. This limitation affects participation and authorship for new contributors. So, the main drawbacks of using CVCS are that it requires a network connection to work on the source code, developers must order to contribute to a project, and a single point of failure is an issue when using one server.

2.2.2 Distributed Version Control Systems (DVCS)

Distributed Version Control Systems (DVCS) were created to overcome the limitations of Centralized Version Control Systems, which enable branching and merging, avoid local VCS operations and allow developers collaboration. Because of the limitations in using a centralized version control system, Open Source Software (OSS) projects today largely adopt DVCS. DVCS is designed to work in two ways: it keeps entire file histories locally on each device and

can also sync local modifications made by users with servers again when necessary so that these modifications can be shared with everyone else. In DVCS developers are able to work separately or together but working on the same project as they have access to all repositories needed for their task; any repository can be cloned from another one so there's no more important repository than others. To provide a new way for versioning software artifacts several Distributed Version Control Systems emerged in the software field such as Mercurial Git Bazaar etc., these tools have been adopted by many Open Source Software (OSS). The operations in DVCS are much faster than those found in CVSS because they're done locally while CVSS operations require remote connection; some consider that distributed systems will soon replace centralized ones because they suit bigger projects with more independent developers who want full functionality even without network connection available, offer advantages like earlier drafts of your work being saved without requiring you releasing them publicly or sharing them with other people etc.; three major advantages offered by use of DVSs include flexibility availability which is related solely to its ability not needing access remotely servers for most operation types plus it's very fast due to its locality-based nature– most actions only need occur at one location unlike CVSCS versions where changes must travel through an often slower internet link before reaching their destination; branching merging is easy when working within DVSs due both high level abstraction language features provided but also lack requirement that user has knowledge about how specific models like Subversion operates making this process easier then would otherwise be possible under traditional model; collaboration between members/individual developer vs server/client differences make up most important feature offered by VSCs since team member may work on source code regardless whether connected centrally or remotely–this allows individual developer do exploratory coding efficiently too. There are challenges introduced by use if DVSs including system backup being difficult cause lack central server makes creating backup virtually impossible meanwhile Pessimistic Locks aren't available either nor does present day software offer good support for binary files. Reasons why transition was made from centralized systems towards decentralized ones: offline capability enabling development portion done independently while maintaining integration into main branch + incremental capabilities providing greater efficiency during exploratory coding efforts.

2.3 Existing Version Control Systems

2.3.1 Local Version Control Systems

Early VCS were intended to track changes for individual files and checked-out files could only be edited locally by one user at a time. They were built on the assumption that all users would log into the same shared Unix host with their own accounts. As you can imagine, these early systems made it easier for small teams to revisit code states from various points in history.

Source Code Control System (SCCS)

SCCS was released in 1972, and it is one of the first successful VCS tools. It was written by Marc Rochkind at Bell Labs, who wanted to solve the problem of tracking file revisions. The tool made it significantly easier to track down bugs introduced into a program. SCCS is worth understanding at a basic level because it helped set up modern VCS tools that developers use today.

Architecture Like most modern VCS tools, SCCS has a set of commands that allow developers to work with versioning of files. The basic command functionality are:

- Check in files to track their history.
- Check out specific file versions for review.
- Check out specific file versions for editing.
- Check in new file versions with comments explaining the changes.
- Revert changes made to a checked out file.
- Basic branching and merging of changes.
- Print a log of a file's version history.

A special type of file called a `s-file` or a `history file` is created when a file is tracked by SCCS. This file is named with the original filename prefixed with a `s.`, and is stored in a subdirectory called `SCCS`. So a file called `test.txt` would get a history file created in the `./SCCS/` directory with a name of `s.test.txt`. When created, the `s-file` contains the original file contents, a header that contains the file's version number, and some other metadata. There are also checksums stored in the `s-file` that are used to verify the

integrity of the file (i.e. to make sure that the file has not been tampered with). The `s-file` content is not encoded or compressed in any way, which is a clear difference from modern VCS tools.

Revision Control System (RCS)

Advantages

Disadvantages

2.3.2 Centralized Version Control Systems

Version control technology continued to evolve, leading to centralized repositories that contained the 'official' versions of their projects. This was good progress, since it allowed multiple users to checkout and work with the code at the same time as well as making commits back into this central repository. Furthermore, network access was required for people who wanted to commit changes they had made locally.

Concurrent Versions System (CVS)

Concurrent Versions System (CVS) was developed in the 1980s by Larry Wall and was the first VCS to become widely used for collaborative software development. Concurrent Versions System is a centralized VCS, meaning that there is a single server that stores the entire history of the project. The server is the only place where files can be added, removed or modified.

Advantages

Disadvantages

Apache Subversion (SVN)

Subversion (SVN) is a centralized VCS that was developed by CollabNet in the year 2000. Subversion was developed to replace CVS, which was becoming increasingly difficult to maintain.

Advantages

Disadvantages

Perforce

Perforce is a centralized VCS that was developed by Perforce Software in 1995. Perforce is a commercial VCS that is used by many large companies, such as Google, Adobe and IBM. It is still one of the largest version control systems in use today.

Advantages

Disadvantages

2.3.3 Distributed Version Control Systems

In a distributed version control system, all copies of the repository are created equal - there is no central copy of the repository. This design principle encourages commits, branches and merges to be created locally without network access and pushed to other repositories as needed.

Git

Git is a distributed VCS that was developed by Linus Torvalds in 2005. Git is a free and open source VCS that is used by many large companies, such as Google, Facebook and Twitter. It is one of the most popular version control systems in use today.

Advantages

Disadvantages

Mercurial

Mercurial is a distributed VCS that was developed by Matt Mackall in 2005. Mercurial was developed with the same goal as Git, to maintain the Linux kernal project.

Advantages

Disadvantages

BitKeeper

Advantages

Disadvantages

Darcs Advanced Revision Control System (Darcs)

Advantages

Disadvantages

Monotone

Advantages

Disadvantages

Bazaar

Advantages

Disadvantages

Fossil

Advantages

Disadvantages

2.4 Summary of Key Features

2.4.1 Repositories

2.4.2 Commits

2.4.3 Branching and Merging

2.4.4 Pulling and Pushing

Chapter 3

Design

Chapter 4

Implementation

Chapter 5

Evaluation

Chapter 6

Conclusion