COMPARISON OF VERSION CONTROL SYSTEM TOOLS

Nasraldeen Alnor Adam Khleel
PhD student, Institute of Information Science, University of Miskolc
3515 Miskolc, Miskolc-Egyetemváros, Hungary, e-mail: nasr.alnor@uni-miskolc.hu

Károly Nehéz
Associate professor, Institute of Information Science, University of Miskolc
3515 Miskolc, Miskolc-Egyetemváros, Hungary, e-mail: aitnehez@uni-miskolc.hu

Abstract

Version control systems (VCS) are widely applied at software companies as a collaborative tool and to maintain multiple versions of source code and documentation. VCS is a software tool that manages development of software projects and provide methods to manage several developers working together and track them. Collaboration considers the master purpose of version control systems. Modern VCS supports the parallel development of artifacts using branches and merges. Currently, the version control system adopts on two approaches to software development, the Centralized Version Control System (CVCS) and the Distributed Version Control System (DVCS). This article introduces the concepts and comparison of Version Control Systems and some criteria to consider when selecting.

Keywords: *Version control systems (VCS), CVCS, DVCS, CVS, Git.*

1. Introduction

Version control systems (VCS) are a system used to stores source code, documentation, and manages the development of software projects [1], by managing modifications and configuration files [5]. VCS is a system that saves modifications done by individual software developers. VCS makes the development process easier and faster, where provides the ability to track and control modifications to data over time. A VCS provides many advantages for software developers, it helps to share data between nodes, and each node can be kept up to date with the latest version of the data [9]. The version control systems are not only related to the modifications of data, but also the reasoning behind the modifications. VCS stores information about which files were modified, when they were modified, who made the modification and what the files contained before the modification [19], as well as help developers to know who works on these files [9]. So, VCS allows the developers to cooperate and work on the same software project at the same time [12].

2. Literature review

This section gives insight into the version control system: Structure of Version Control System, Purposes for Version Control systems, Types of Version Control System, and Version Control System tools.

2.1. Structure of the version control system

A version control system provides a basic principle and way of storing files and modifications done, this is achieved by using a repository. The repository includes the most recent version of each file and the modification history that led to this stage, each modifies usually contains additional information such as the author and a short description and a modification time. VCS can be divided into centralized and distributed version control systems by studying the method they deal with repositories and the sharing of modifications between contributors. The files and all the information associated with each version are stored in a repository. There are many repository models popular employed in a Version Control system. The earlier model of Version Control systems has a local repository, modifications will be done on the repository that stored on the same machine. Client-server models use a central repository that allows all clients to read and submit modifications. In recent years, the distributed repository has gained increased attention because it allows cooperation without the need for a central repository. In the past twenty years have seen the emergence of distributed version control system (DVCS), these systems have many repositories, each works independently and still have a master repository [9].

2.2. Purposes for version control system

Collaboration considers the master purpose of version control systems. Version control systems have appeared in the first release since the 1970s - with the principle of easier management of source code files that have been continuously modified by many software developers. These systems provide the ability to see the evolution of data over time, a snapshot of it at a specific time, or a method to recover-restore if necessary. These factors made version control systems a vital component of collaborative systems [9]. Older VCS like CVS do not support renaming trace files at all and new files start with new archives unless the inventory is manually modified, while most new systems support file renaming [6]. Modern VCS support the parallel development of artifacts using branches and merging, maintain a major branch to represent current development works, create new branches to the master branch to represent the released versions, and track bug fixes in the released product [23]. There are many advantages of the version control system: VCS has been proven to accelerate and simplify the software development process. VCS allows and helps people to work freely with the team, where they can work on any file at any time without overlapping each other's work by writing over other people's code since VCS enables collaboration so people can share the source code more easily. Every time people commit modifications, they create a new version of the corresponding file. VCS professionally keeps the version. Also, VCS allows the older version of the source code to be safely stored in the VCS repository. A VCS can be extremely useful because it will allow people to recover from accidental deletions or edits [1]. In other words, the core functionality of VCS: made backing up for source code, allow developers to collaborate and work together [12].

2.3. Type of version control system

Version control systems provide a method to manage several developers working together and track that. Over the ages, there was an evolution of Version Control Systems [8]. There were two approaches to version control system: Centralized and Distributed - Version Control System. Both approaches are in use to a large extent today, although centralized version control is the most common. The centralized model work based on the client-server model as well as lets users work at a single central repository. While DVCS is a distributed model that provides a central repository for every user [1]. The main difference between approaches of version control systems is that the DVCS do not need a central server to store the repository as it's like in CVCS, every user or developer has a complete repository on their local computer. Developers can collaborate directly without needing central authority or incurring central administration overhead, and the acts of snapshotting modifications and publishing modifications can be decoupled [4].

2.3.1. Centralized version control systems(CVCS)

A centralized Version Control System developed to overcome the issues faced by developers when they need to work with the other developers on the same systems [8]. This system enables the developers to work cooperatively, where stores the main copy of files history and keep track of files and save all of the information in the local repository. It is one of the easy and simple forms of version control, save all the modifications to the files under revision control in a database [21]. CVCS are called centralized because there is only one central server or repository holding the version database where the developers check out their projects on their local computers [19]. The server can be accessed via the network [4]. The server maintains a complete record of issues, while clients only maintain a local copy of the shared documents, all the developers make their modifications on repository through checkout but only the last version of the files is retrieved from the server, it means that any modifications made will automatically share with other developers [1]. 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 [7]. 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, if central server 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 corrupted, everything will be lost [8]. 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, release modifications with full revision history, etc. This limitation affects participation and authorship for new contributors. So, the main drawbacks to use CVCS: require a network connection to work on the source code, developers must order to contribute to a project, a single point of failure is an issue when using one server [2].

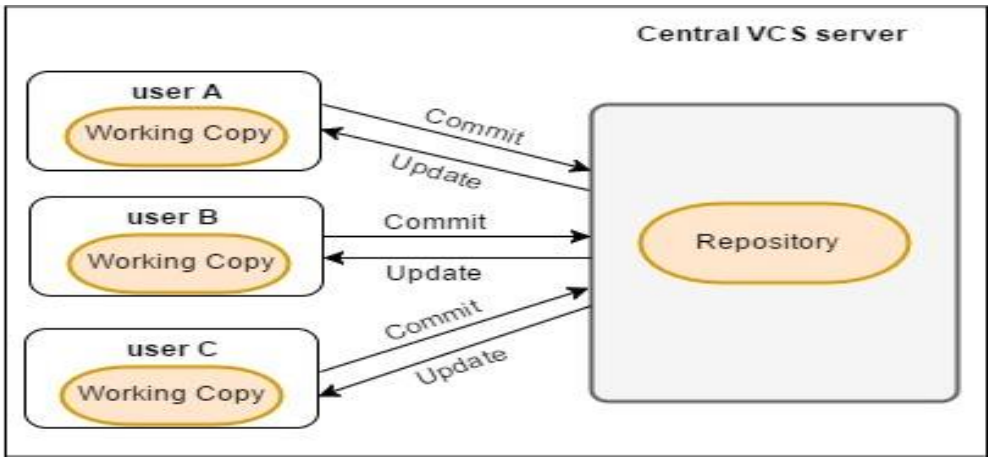

Figure 1. *Centralized Version Control Systems*

2.3.2. Distributed version control system(DVCS)

Distributed Version Control Systems (DVCS) developed to overcome CVCS restrictions, to enabling branching and merging, avoid the local VCS operations, and allow developers collaboration [12]. Because of the limitations of using the centralized version control system, Open Source Software (OSS) projects these days largely adopt the DVCS. DVCS is designed to work in two ways as it keeps the entire file history on each device locally and can also sync local modifications the user made with the server again when necessary so that the modifications can be shared with the whole team [1]. In DVCS the developers can work with different groups of people in different ways working in the same project [8]. As well as any repository can be cloned, and, from a conceptual point of view, no repository is more significant than any other. In practice, the development team will organize the repositories in the hierarchy and at least one of the repositories will be marked as the central repository [20]. To provide a new method for versioning software artifacts, several Distributed Version Control Systems emerged in the software field such as Mercurial, Git, and Bazaar, these tools have been adopted by many Open Source Software (OSS) [2]. The operations in DVCS are much faster than operations in CVCS because they are local [4, 12]. DVCS considered being the future of version control systems, because it suited for huge projects with more independent developers, and provides important advantages by allowing users to work and use a complete version control feature set even when there is no network connection. DVCS allows version control of modifications done locally allowing early drafts of work to be revision without requiring it to be released to others [9, 11]. Three are many advantages of using a distributed version control system: flexibility, hosting services like Github [20], availability, it is very fast due to its local nature to the majority of operations, it doesn't require access to remote servers, and, branching and merging can be done very easily in DVCS [8]. Collaboration between team members and allow individual developers to be servers or clients are the most important features offer by version control systems, so developers can work on source code without being connected to a central or remote repository [2, 18]. There are some Challenges Introduced by DVCS: it lacks an understandable version numbering system, where there is no centralized versioning server, and use hash modifications or a unique GUID. So, the lack of a central server makes system backup so difficult. The two most popular complaints about the disadvantages of DVCS are that: pessimistic locks are not available, and they have weak tools for binary [9]. The reasons for the transition from centralized to decentralized version Control Systems: the ability to work offline and the ability to work incrementally. The ability of developers to made several roles, such as developing a new task or fixing errors, and the ability to do exploratory coding efficiently [12].

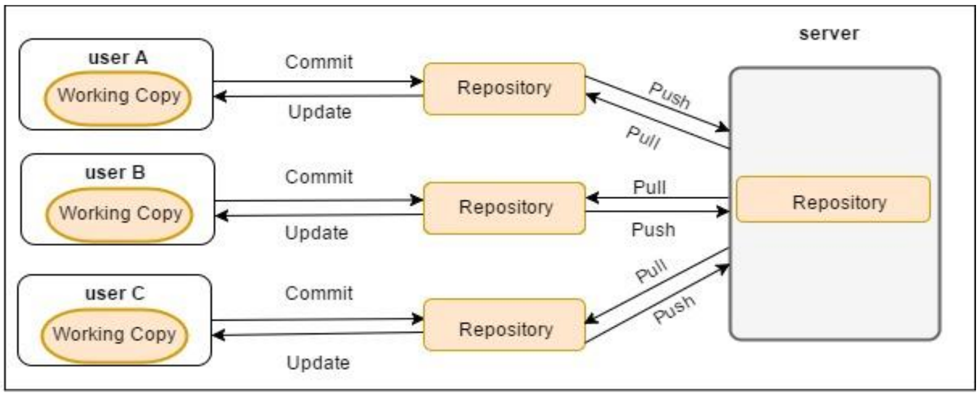

Figure 2. *Distributed Version Control Systems*

3. Version control system tools

Version control system (VCS), besides other tools such as the issue tracking system, is central to how development work is organized for several open software projects. The version control system tools decide how easily people can contribute to the project, how to coordinate the development of new features, and how to integrate development lines, how code is reviewed, and already released. The new generation to the version control system solved some limitations of the previous generation of the version control system [23]. Version control systems are popular tools that support parallel work over shared projects and offer support for synchronization of parallel modifications on those projects [7]. CVS and Subversion are the most used CVCS tools, but several open source and closed projects have been moved to DVC since the emergence of DVC tools such as Git, Mercurial, Bazaar, Perforce, and BitKeeper [1]. The most common tools are Bazaar, Git, and Mercurial, these tools enhance the merging and branching features. There are different criteria to choose version control system tools, such as reliability, adaptability, usability, extensibility, and integration [14].

3.1. Concurrent version system(CVS)

Concurrent Versions System (CVS) was the first popular central version control system for collaborative work. CVS is open-source software for version control, where uses a client-server architecture, the server is responsible to store the current project and its history. Written by Dick Grune in 1986, it is the most well-known exponent on CVC. It was designed and coded based on the Revision Control System (RCS) that which only was allowed to manage individual files rather than projects. CVS has a central repository, where the users make a local copy project from the current project [17]. VCS allows clients connected to the server to "Check out" a full copy of the project, work on this copy, and later "Check-in" their modifications [16]. CVS was one of the first VCS introduced the concepts of branches and feasibility features that exist today in almost every VCS [4]. It can be integrated easily with other authentication servers like Active Directory. It is user-friendly and is based on a client-server architecture. CVS servers run on all major platforms – Windows, UNIX, Linux, etc. [13].

3.2. Apache subversion(SVN)

Because of the limitations of CVS, in the year 2000, CollabNet decided to write a system that is like CVS but without problems [4]. SVN is an open-source version control system and abbreviation of Apache Subversion [16] SVN developed as CVS replacement with some enhancements, considers the final step in the evolution of centralized version control, and it is sharing several advantages with Perforce [6, 13]. CVCS such as Subversion rely on a client-server architecture, the server saves a full history of versions while clients save only a local copy of the shared documents [11]. Recent versions of Subversion have better branching and merging abilities than the earlier versions [14]. FSFS and Berkeley DB are two technologies available in SVN for a file system: both of these technologies allow important features such as data integrity, atomic writing, recoverability, and backups if something goes wrong during a commit operation, all the process is canceled and the repository remains in a stable state, so when committing, everything is transferred correctly or none [17]. Subversion is an effort to overcome the main CVS limitations, it can rename and make copies of files inside the repository because it does not use an RCS-like file system, which also allows for more efficient binary file processing. Instead, SVN tracks all modifications, not only in every file but also in the complete directory tree. Besides, SVN keeps an invisible hash table for each file or directory allowing the user to write their information about them such as permissions, MIME types, owner, etc. [17].

3.3. Perforce

Perforce was founded in 1995 by Christopher Sewald with a commitment to producing SCM programs that are committed to performance and reliability. Perforce today is one of the biggest and most common source code management systems for customers using Perforce for source code and digital asset management client-server, Perforce provides a variety of features to organize code, define where components are and control deployment of components for reuse, so Perforce is a highly versatile, flexible, powerful SCM system, and allows all kinds of warehouse structures [15]. Perforce provides command-line clients and visual clients to run, Visual clients include tools for embedding and tools for tracking debugging, as well as supports the common functionality for version control such as check-ins and checkouts, branching, merging, and tagging [6].

3.4. Bazaar

Bazaar tool focuses on the simple and easy user interface, the user interface supports some popular workflows like checkouts, and it is a better substitutional to all developers using pushing to a central repository [6]. The bazaar is a distributed version control system such as Mercurial and Git, each developer works on his local branch independent of others, this means that the developer has to push his modifications to the server after committing them locally if many developers are working on the same project the recurring task is to merge the modifications. There are two different approaches to use Bazaar repositories: checkout and branches [10].

3.5. Git

Distributed Version Control Systems such as Git is the best choice for project development, but the researches done in the past haven't given light to the enhancement of the architecture and methodology of Git [22]. Git developed in April 2005 as an open-source code and distributed version control system, to handle source code with speed and efficiency [16]. Git is considered today one of the most popular source code management tools for the huge number of OSS projects [3]. The main purpose of Git is distributed revision control. Git allows cloning the entire repository it is possible to work without internet connection [16]. Git became very widespread in the open-source community because it supports parallel collaborative work, and offers some very interesting advantages, such as resolving conflicts that arise during synchronization of parallel but that is modifications take time-consuming [7]. An important attribute of Git is the method it handles the time of modifications in the source code, it tracks only the time of commit, and also support branches, and this allows developers to create several lines of development and merging them [19]. One of the most important differences between Git and other VCS is the way how Git treats data. In Git, instead of keeping a reference to the modifications made to files, every time a user commits a new version, Git takes a snapshot of the state of all files, besides the advantages of working with a local repository [17, 18]. The properties of Git: Git does not use delta encoding to store files, it stores snapshots of all files in a tree structure, Git tracks content, not files [4].

3.6. Mercurial

Mercurial is a program that developed at the same time when Git was released. Mercurial was designed and developed with the same drive as Git, which is the keeping of the Linux kernel project [16]. Mercurial provides distributed version control system features, with ease of use and good documentation. Installation may be a bit cumbersome due to dependencies on other packages required, Mercurial uses Python [14].

4. Summary

Software development teams use the version control system to manage their projects. VCS record history information about who made commits, how many les the commits modify, when branches are forked or merged, and in some cases when conflicts occur. VCS is designed to store, compare, and manage many versions of source code files [14]. VCS accelerates and simplifies the software development process and enables new workflows, keeps track of files and history, and have a model for concurrent access [4] as well VCS provides many advantages for tracking and controlling the modifications, such as commit, rebase, revert, branch, merge, and log [5]. This work compared the version control system tools depending on many criteria such as file repositories, working directories, and commits, branch and merge, push and pull.

4.1. File repositories

The centralized version control system having a central server, so it needs to have someone manage the server, In the event of a breakdown of the server or network, developers will be unable to do their works. The distributed version control system does not have a central server [14]. In DVC, each team member owns the full repository on the local development machine called the local repository, the developers can copy the remote repository into their local repositories and commit to the local repository [19]. In DVC, backups are taken to the server repository, guarantee that all committed modifications are considered during the backup process. In DVCS, every client has their repository and there is no guarantee the nodes have all the modifications [9]. Another important issue of DVCS is that locking behavior is not possible. In a centralized system, could implement a locking policy for the main repository, locking a file when a user is working on it until he makes a check-in of the file [17].

4.2. Working directories and commits

In a centralized version control system environment to generate any modification to the repository's history, we must enter to the central server. A centralized version control system environment grants anyone to modify any a portion of their native repositories past events [14]. In DVCS, file modifications are organized in commits, where the information about the commit time and the branching is preserved, branching is used to separate commit modifications from each other [19], CVCS have similar workflows. In contrast, DVCS is very flexible and one can adjust the setup of DVCS to fit one's needs [4]. In DVCS the files are not modified directly in the repository, working copy is created with the checkout command, the modifications a user makes are transferred to the repository with the commit command [14]. In Git every team member has own local repository and working directory to make commits, the local repository allows the member to privately work, since team members do work on the same project, they need to share and synchronize their work, so each team member must pull other member's modifications from the central repository and, push his modifications to a central repository [18]. Besides, when a commit implementation, the data is not lost, which means that every action can be undone, making impossible to make a catastrophic error [17].

4.3. Branch and merge

Merge consider one of the most important operations in the version control system, but merge process may lead to conflicts, if conflicts happen, the merge process is abandoned until the conflicts are solved [17]. There is a difference between merging in DVCS and CVCS, some CVCS tools do not explicitly track merge information, such as conflicts and their resolution, so for several merges between the same branches, the correct revisions numbers need to be specified manually, when a merge conflict arises, git writes the resolution to its commit. Additionally, the merge that occurs when the SVN developer updates its work copy is not recorded. DVCS stores sufficient information to support branching and merging, rather than being directly developed against the mainline, there are no enforced master branches in DVCS because each DVCS repository is a complete repository [23]. Git performs merging in two different ways, first, if the commitment object that the main branch refers to is a direct ancestor to the compliance object that the branch referred to, Git quickly fuses forward, in the other case, if the branch deviates from an older point, Git performs a merge [17].

4.4. Push and pull

The early version control system was "push" systems, this means that developers push their code modifications into the repository when they're ready, this works well when we can trust all programmers to always do reasonable things and put a good code in the repository. as well there are "pull" systems, each developer has copies of a repository, which are branches, the modifications only go between these branches when someone pulls the modifications from someone else’s repository to their possession. Git was originally only a "pull" system, but recent versions also support the "push" function, many projects in Git disable the "push" function and manage the project as a "pull" system. Mercurial has "push" and "check out" functionality by default, although it's somewhat easy to disable [14].

5. Conclusion

Nowadays all software development teams must use a version control system to manage their projects. Version control systems record history information about who made commits, when branches are forked or merged and, in some cases, when conflicts occur. Version control system tools determine how easily people can contribute to the project, how to coordinate the development of new features, and how to integrate development code lines, how code is reviewed, and already released. CVS and Subversion are the two most used tools in CVCS, but since the appearance of DVCS tools such as Git, Mercurial, and Bazaar, several open-source projects moved their source code repositories from CVCS to DVCS. Git is the most common and widely used tool among DVCS. This paper discusses the concepts and Comparison of Version Control System tools.

References

[1] Zolkifli, Nurlisa, N., Ngah, A., Deraman, A.: Version Control System: A Review. Procedia Computer Science 135 pp: 408-415. 2018. **https://doi.org/10.1016/j.procs.2018.08.191**

[2] Rodríguez-Bustos, C., Aponte, J.: How distributed version control systems impact open source software projects. 2012 9th IEEE Working Conference on Mining Software Repositories (MSR). IEEE, 2012. **https://doi.org/10.1109/MSR.2012.6224297**

[3] Lee, H., Seo, B.-K., and Seo, E.: A git source repository analysis tool based on a novel branch-oriented approach. 2013 International Conference on Information Science and Applications (ICISA). IEEE, 2013. **https://doi.org/10.1109/ICISA.2013.6579457**

[4] Stefan, O.: Version control systems. Computer Systems and Telematics pp: 11-13. 2009.
[5] Bo, C., and Curtmola, R.: Auditable Version Control Systems. NDSS. 2014. **https://doi.org/10.14722/ndss.2014.23184**
[6] Baudiš, P.: Current concepts in version control systems. arXiv preprint arXiv:1405.3496, 2014.
[7] Hoai, L.N., Ignat, C.-L.: An Analysis of Merge Conflicts and Resolutions in Git-Based Open Source Projects. Computer Supported Cooperative Work (CSCW) 27.3-6 pp: 741-765, 2018. **https://doi.org/10.1007/s10606-018-9323-3**
[8] Bhoir, Pratik P., Patil, H.: Evolution of Version Control Systems and a study on TortoiseSVN. Evolution 5.06, 2018.
[9] Ali, K., Tansel, A.U.: A survey of version control systems. ICEME 2011 (2011).
[10] http://docplayer.net/134726273-A-quick-guide-to-bazaar-for-ccp4-developers.html.
[11] Nguyen, H.L., Ignat, C.-L.: Parallelism and conflicting changes in Git version control systems. IWCES’17 - The Fifteenth International Workshop on Collaborative Editing Systems, 2017.
[12] Kıvanç, M., et al.: Transition from centralized to decentralized version control systems: A case study on reasons, barriers, and outcomes. Proceedings of the 36th international conference on software engineering. 2014. **https://doi.org/10.1145/2568225.2568284**
[13] Rao, N. Rama, Sekharaiah, K.C.: A Methodological Review Based Version Control System with Evolutionary Research for Software Processes. Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies. 2016. **https://doi.org/10.1145/2905055.2905072**
[14] https://ianclatworthy.wordpress.com/2007/06/21/version-control-the-future-is-adaptive/ .
[15] Jo, W.: Managing Software Reuse with Perforce. Mandarin Consulting.
[16] Pranjal, G., Budhkar, S.: Review on Version Control with Git. (2018).
[17] Escobar, G.J.A.: Software Development and Collaboration: Version Control Systems and Other Approaches. 2011.
[18] Tepavac, I., et al.: Sustavi za verzioniranje, alati i dobra praksa: slučaj Git.
[19] Jasmin, R., Wagner, S.: Which change sets in Git repositories are related?. 2016 IEEE Internati-onal Conference on Software Quality, Reliability and Security (QRS). IEEE, 2016. **https://doi.org/10.1109/QRS.2016.52**
[20] German, Daniel M., Adams, B., Hassan, A.E.: Continuously mining distributed version control systems: an empirical study of how Linux uses Git. Empirical Software Engineering 21.1 pp: 260-299. 2016. **https://doi.org/10.1007/s10664-014-9356-2**
[21] https://www.lexjansen.com/phuse/2019/tt/TT02.pdf.
[22] Rana, M., et al.: Source code management using version control system. 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO). IEEE, 2017. **https://doi.org/10.1109/ICRITO.2017.8342438**
[23] De Alwis, B., Sillito, J.: Why are software projects moving from centralized to decentralized version control systems?. 2009 ICSE Workshop on Cooperative and Human Aspects on Softwa-re Engineering. IEEE, 2009. **https://doi.org/10.1109/CHASE.2009.5071408**