Lab 1: Software Version Control

ANLY 560 Spring – Functional Programming methods for Analytics

Yufeng Fan

Jan 24, 2021

Table of Contents

The Purpose of Software Version Control

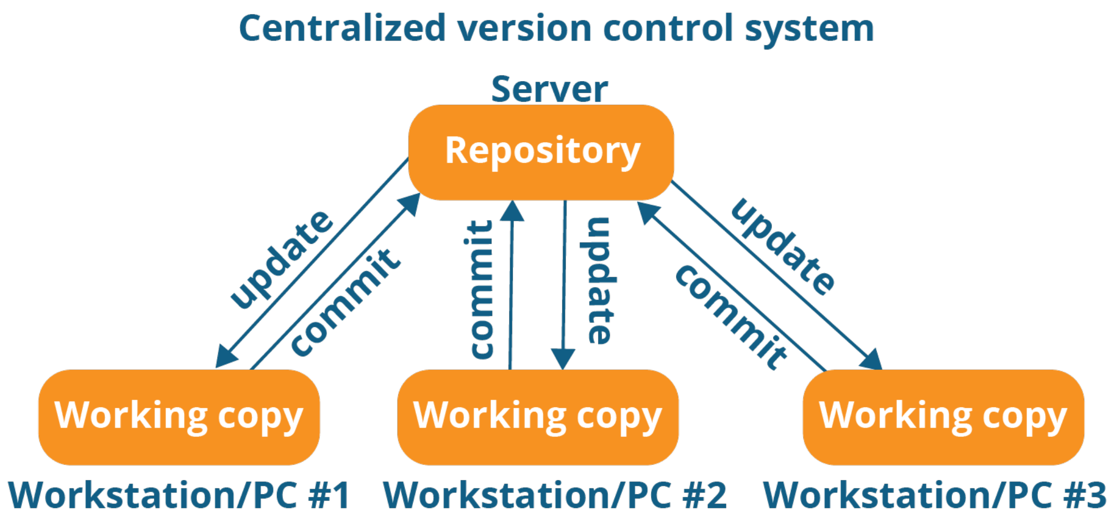
The world of Software Development has seen great changes and improvements over the past few decades. People are always demanding software applications to achieve better performance and greater scalability to make greater impact, as well as hoping that software can be built and deployed in a much shorter period. Given the fact that the entire world is accelerating, people have created many modern software development methodologies, frameworks, and models to guide software development teams to work not just in a faster, but in more efficient and more reliable ways too. Software Version Control (SVC) System was one of excellent frameworks created that enables development teams to work efficiently but also ensures great reliability to handle today's fast paced world that constantly requiring turnarounds and change of requirements. There are three main areas that an SVC System helped development teams. Better collaboration, better software management and great traceability. I will discuss each of them in-depth in the following sections to illustrate what are the features that an SVC System provides that produced those benefits.

Better Collaboration

First and foremost, to ensure better collaboration and allow developers to move efficiently, most SVC System can provide branching capabilities that allows development teams to work on different parts of a project at a large scale without impacting on each other.

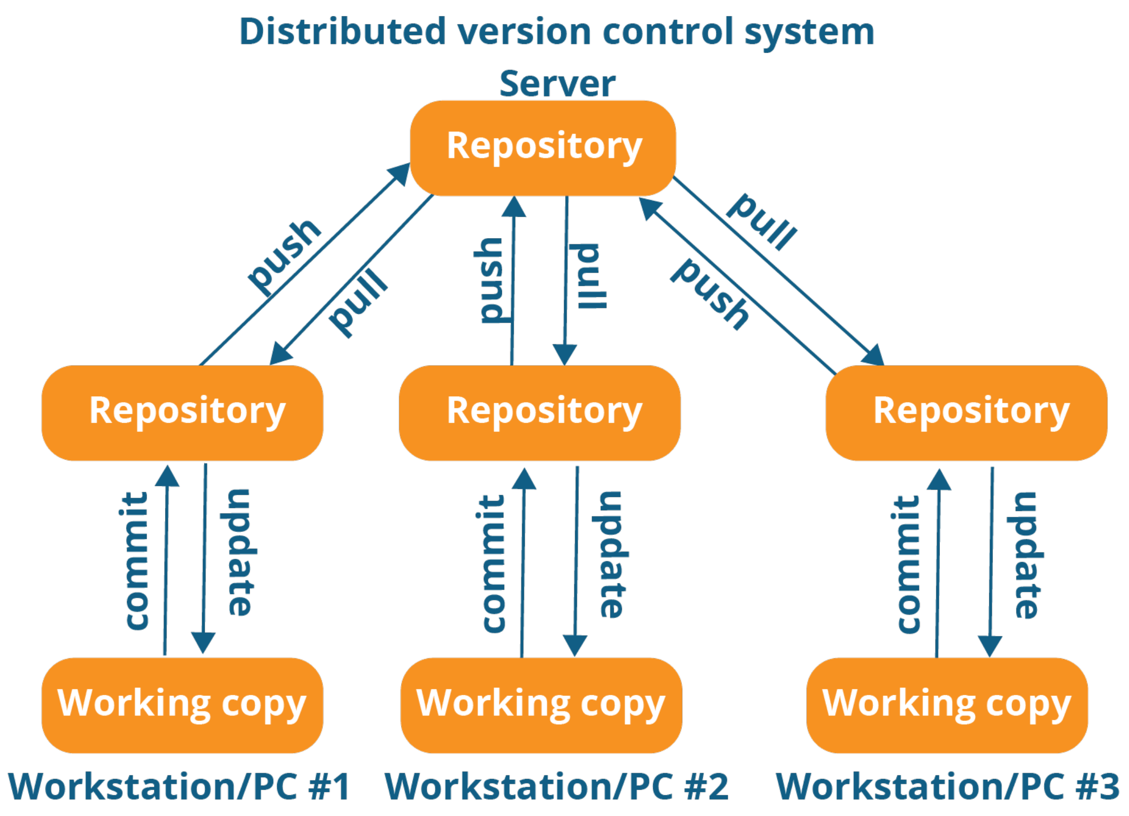
Centralized Version Control System

Two are two main components in a centralized source control system, a server, and a client. The server often represents the master copy of the repository and contains all the versions of the code. And often, pieces of the code under working are locked. Which means only one developer can work on that part of the code at a time. Access to the code base are often controlled by the server so that when developers checks out the code, that piece is locked, and the lock will get released when their code checks back in. How does it relate to versioning? So, when an element is check in, a new software version of that piece of code will be created. Here is a diagram shows this client and server model in centralized version control system.



Distributed Version Control System

Unlike the centralized version control system, where only one can work on a piece of the software at a time, the distributed version control system doesn't have that barrier because there is a master copy of the code base on each of the client side, rather than on a server. So that each client would has their own server as well as a copy of the entire history of the code. One huge benefit is that every client can work locally in a "off-line" mode. And that is why it is called distributed. And the client will issue a request to the owner of the master copy when they are trying to merge their local copies into the master copy. Here is another picture shows the client and server model in distributed version control system.



Pull Request and Merging

While working on a specific part in a working copy, there is a demand to allow contributors to further divide work into smaller chunks and workable unit. Hence there is another concept of commits that represents a smallest unit of work submit to the SVC, and further there is a Pull Requests concept, AKA code review process, to ensure the work will be peer-reviewed and tested by a verification process before merge the work back to the branch.

Better Software Version Management

Secondly, to ensure better software version management, we need a versioning system that can assign a unique identifier to uniquely link states of the project, it could be a main milestone completed or could be a small bug fix. So that during later time, contributors could easily jump between different versions of the software to restore and recreate the state of the world through a simple version number. Often an SVC system has features of tagging versions to branches and commits to help developers better manage the project.

Semantic Versioning

One popular versioning principle is called Semantic versioning (SemVer) created as a universal way of defining states of software as new features, libraries, bug fixes and tests are being built every single day. Since the goal of SemVer is to bring sanity into the rapid software development world. The creator of SemVer proppose a simple set of rules to guide how to determine version numbers and how they should get incremented, and we use the version numbers to communicate changes in the software easily. The subject of the version is often APIs in the software, we define format of such versions as X.Y.Z (Major.Minor.Patch). Patch version get incremented by bug fixes not affecting the API, backwards compatible API additions/changes increment the minor version, and major version were driven by backwards incompatible API changes.

After software been properly version tagged, we need the SVC system to able to work with release management system to ship software on different versions to different environments.

Better Traceability

Finally, to ensure great Traceability, the SVC System ought to provide transparency on code ownership. The SVC system would trace every single line of code and record the ownership through commits and Pull requests. So, it’s very transparent and easy to see when, and who made the change. As far as to bridge between code and requirement, SVC systems has APIs to integrate with project ticket management tools, like Bitbucket and Jira, to connect each commit to their project development ticket and story.

The Adverse Impact on Software Development If Without Software Version Control

After discussed with all the benefits that SVC system would bring to us, I believe it's safe to say it would be catastrophic for a software development team does not use a SVC system in their software development process.

Hard to Collaborate

Adopting SVC system was not making things complicated. On the contrary, it alleviates efficiency in communication between members of the team. If there were no SVC in Software Development, each team could only allow one member making one change at a time. And the rest of the team member could either wait or work on other projects. Which means collaboration will be extremely hard.

Poor Software Version Management

One might argue that not adopting SVC system in a team environment might be unwise, how about in a one-man team? I believe the answer is the same. Even in an environment that doesn't require collaboration, an SVC system could still bring enormous benefits in software version management. Failed to use such system would make it difficult to archive progress and record a milestone as well as state of the software when it's needed. Not to mention the ease of jumping between different copies and versions of using an SVC system, it would be impossible to even take a high-level view of all the versions available without using a SVC system. I would only imagine having many local copies that are poorly arranged in the file system and it would rapidly go into chaos if not having such system in-hand.

Poor Traceability

An often-ignored benefit of an SVC system is to provide traceability. Not having the annotated history of the code at your fingertips when reading the code would be a nightmare experience in software development, because without such features, it would be extremely difficult to understand and describes the purpose and intention of such change. And in-turn, it would be hard to figure out the root case analysis. Hence, it would be hard not only for future developers, but also would be hard for current contributors to debug and maintain the code.

Conclusion

After both arguing the purpose of adopting SVC system and potential adverse impact on without having such system, I would like to emphasis that, even it's possible to develop software without using a SVC system, however, doing so would hugely impair software development experience and would make teams to spend more time in managing messy code and logistics rather than spending majority effort in developing code to provide more business values. And doing so subjects the project to a huge risk that no management officers nor professionals would be advised to accept. Hence given the stated benefits of having an SVC system. I would strongly recommend any software development teams to start exploring and adopt one system that suits their own best needs and use this tool to make software development an efficient and smooth experience.

Reference

https://www.atlassian.com/git/tutorials/what-is-version-control#:~:text=non%2Dsoftware%20projects.-,Benefits%20of%20version%20control%20systems,scales%20to%20include%20more%20developers.

https://www.git-tower.com/learn/git/ebook/en/command-line/basics/why-use-version-control/

https://www.perforce.com/blog/vcs/branching-definition-what-branch

https://sifterapp.com/academy/start/foundation/

https://www.geeksforgeeks.org/centralized-vs-distributed-version-control-which-one-should-we-choose/

https://semver.org/

https://medium.com/faun/centralized-vs-distributed-version-control-systems-a135091299f0