July 25, 2018

What Is DVCS Anyway?

GIT AT SCALE | VERSION CONTROL

By Chuck Gehman

What Is a Distributed Version Control System?

> A distributed version control system (DVCS) is a type of version control where the complete codebase — including its full version history — is mirrored on every developer's computer.

Changes to files are tracked between computers. For example, my workstation and yours. In the beginning, this required specific coordination strategies to maintain consistency in projects, so all the developers could keep track of what was happening to files at any given time.

The cited advantages are:

- Branching and merging can happen automatically and quickly
- Developers have the ability to work offline
- Multiple copies of the software eliminate reliance on a single backup

Another cited benefit is the increase in developer productivity. Because all the code is on your own workstation, it makes common activities quick: check in, check out, and commit. This was vital because back in the "olden days," server access, and even workstations, were slower than today.

> It was a great idea. Many innovations have sprung from the collaborative way open source DVCS brought developers together.

Related Blog: What Is Version Control?

Git Made DVCS Popular

The Git DVCS model is applauded for allowing developers to work independently and disconnected, and to experiment freely. But, does this model (as it's classically defined) have a future in accelerated development? Especially with high-performance enterprises needing to deliver faster and at better quality?

The enterprise has been searching for an optimal developer solution that isn't technology based on a peer-to-peer model. Though not everyone realizes it — or wants to believe it — DVCS is not how large organizations run.

Commercial organizations' software spans multiple repositories. It includes large binaries and artifacts. Storing entire repos on local workstations, as DVCS demands, gives organizations security and compliance risks. The environment also creates an explosion of code and artifacts that damages speed and productivity.

So you are probably looking for a solution that fully embraces developer desires but also meets your needs. Because let's be honest. If you are using a centralized server for CI/CD builds and backups, are you really using DVCS anyway?

The Evolution of Distributed Version Control Systems

The nature of software development activities has changed a lot since 2005. That was when distributed version control began its climb in popularity. The original Linux hackers drove the need for a new, free version control system to support their work on the kernel. It was great because developers were working mostly from home and contributing to the same project.

Today, there are far more challenging requirements. An enterprise has numerous projects going at the same time, each potentially made up of thousands of files. It is an explosion of

code. Projects often include code and non-code assets, artifacts, containers, and even graphics, audio, movies, and other binaries. Distributed version control systems just can't keep up with the demands.

Speed and Security Shortfalls

Let's start with performance. It is no longer a handful of developers working from home on a project. Now, it can be hundreds or thousands of contributors. For example, Android OS code consists of upwards of 1,100 repositories. Continuous Integration (CI) builds can take forever on DVCS. And the result — developers spending time waiting for pass/fail results — is costly.

Security is now a far greater concern than it used to be. Developers downloading an entire code base onto a laptop is problematic even without the risk of attacks, which are increasingly prevalent. From a security standpoint, you want to follow the principle of least privilege with all your systems, and this includes the version control system. This is cumbersome or impossible to do, when the unit of granularity is an entire repo.

Redundant Backup Plans

When it comes to backups, the redundancy of having your code on individual workstations isn't going to help today. What you need is a single source of truth that is backed up in a way to ensure that you can maintain developer productivity without interruption. That doesn't mean resuming work off someone else's workstation copy of the software. It means defining a recovery point objective (RPO) and a recovery time objective (RTO) in your business continuity and disaster recovery (BC/DR) plan.

Limitations of the DVCS Model

Because of all these limitations and the needs of larger organizations, DVCS models have become the very thing they intended to replace. It's back to the future. Now, the most popular use case for DVCS is to have a central repository as an intermediary between workstations.

For large enterprises, it is difficult to rely on DVCS. Instead of a team exchanging files with one another, on a peer-to-peer basis, they are exchanging huge repos with a central server, pushing and pulling them.

Remember when we talked about the need to manage collaboration between developers, files, and projects? Now, the central repository "hub" holds revised file versions, and we all sync up for consistency purposes.

Ironically, even in the Linux hacker days, the central repository was Linus Torvalds himself. He needed to control what became the "single source of truth" for the project to move forward effectively.

The "Hub" Centralizes Version Control

Let's face it, when we talk about this "hub" or DVCS, we are talking about Git. But there's a problem: Git's basic functionality is not optimal for acting as a central server. There is no specific fundamental technology designed to make it work at scale.

Very good tools have been built on top to manage this issue. They have attempted to provide collaboration and repo management with added features, but the underlying architecture is still the peer-to-peer design from 2005.

Git has become almost entirely a hub and spoke model with enterprise instances that are sold commercially, rather than freely downloaded open source. While wrapped in nice-looking web-based user interfaces with some workflow offerings, compliance and governance are all but absent, and performance at scale is elusive.

It's like Spotify decided to use the code from Napster as their server.

Smarter Way to Control Your Code

A better way to serve the needs of large distributed teams is to use a centralized version control system. They are designed with the ability to deploy multiple high-performance servers that work together to address the needs of a large global workforce.

The cloud and virtualization architectures of today were only in the dreams of computer scientists in 2005. Whether cloud-based or running on an enterprise VM cluster, it has become easy to spin up new servers. All you need is a template of the version control server, and you can scale across your enterprise.

Helix Core is Perforce's version control solution that provides superior performance over DVCS in almost all metrics: scale, traceability, security, and code management. It serves your Continuous Integration/Continuous Deployment (CI/CD) environment with flexible workflows, and it improves your build times dramatically when compared to most other VCS solutions. And, when Helix Core is used with Helix4Git, developers can continue to use their Git-based workflows transparently against a purpose-built enterprise server.

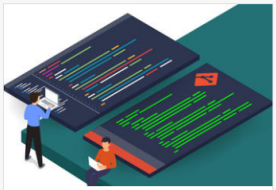
HOW HELIX4GIT WORKS

Chuck Gehman

Technical Marketing Engineer, Perforce Software

Chuck (@charlesgehman) is a Technical Marketing Engineer at Perforce. He has worked as a CTO, architect, developer, and product leader in startups and large enterprises. Chuck has been a member of the IEEE for 20 years, and he's an AWS Certified Solution Architect and Certified Developer-Associate. When he has spare time, he enjoys volunteering for technology education initiatives, attending Meetups, and writing.

Recommended Posts



October 17, 2019

How to Improve Your Git Code Review Workflow

GIT AT SCALE, CODING BEST PRACTICES

October 7, 2019

How to Use Git Shallow Clone to Improve Performance

GIT AT SCALE, VERSION CONTROL

September 27, 2019

How to Use Open Source Code in Proprietary Software

VERSION CONTROL