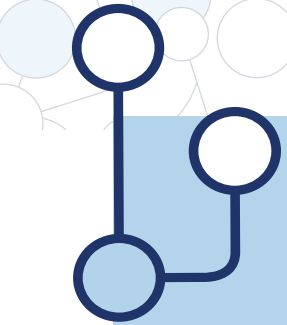
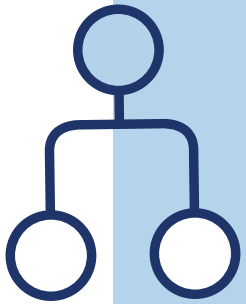
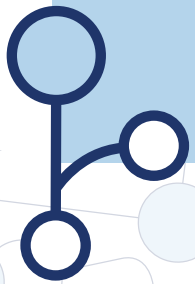
5 WAYS TO SCALE GIT

FOR ENTERPRISE DEVOPS

PERFORCE







Introduction

Git began as a tool to manage source control for the Linux kernel. Today it's the de facto standard among software developers. In fact, its ubiquity is the reason developers have grown to depend on it for code collaboration, knowing that Git offers easy access to all their projects from virtually any platform.

Unfortunately, Git was never intended to scale quite so massively. Enterprises now face the daunting task of balancing developer needs against the implementation of Continuous Integration (CI)/Continuous Testing (CT)/Continuous Delivery (CD) for better, faster, higher-quality releases.

If you're currently responsible for driving your DevOps pipeline, you probably don't care about any of that. You just want things to work and meet your needs. You must be able to:

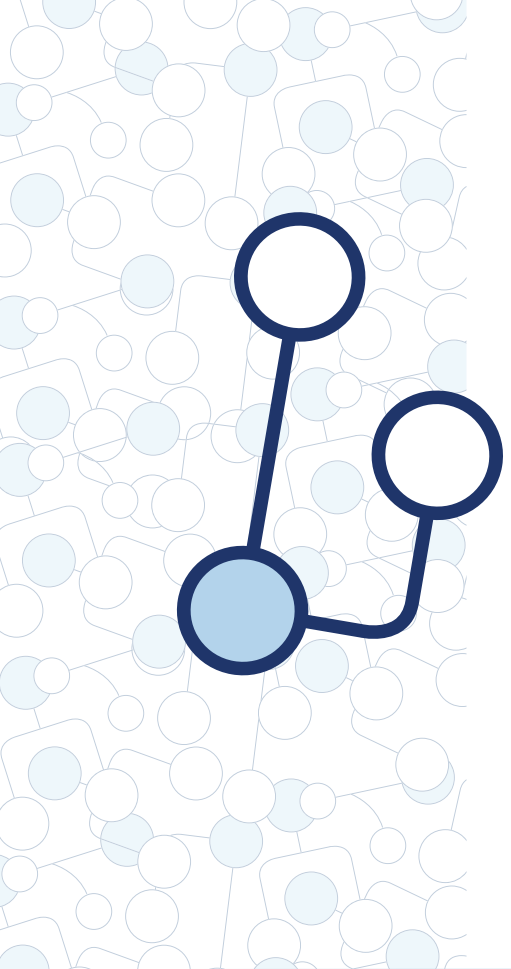
- Simplify and unify change management across all DevOps processes.
- Version all your build artifacts, code, and binaries – preferably in the same place.
- Document, trace, and audit from end to end, to identify vulnerabilities and defects.
- And do it all quickly, faster than the competition as a bare minimum, while keeping your developers happy along the way.

Accomplishing this with Git, especially Git alone, presents the enterprise with challenges that threaten the entire DevOps pipeline. So, how do you overcome the challenges Git poses to enterprise DevOps initiatives?

Custom large-scale tooling is an option usually reserved for the tech giants of the world, those with the resources to dedicate entire teams to building and maintaining their custom solutions. For most organizations, however, it is more cost effective to satisfy developer needs with existing tools that allow the organization to overcome Git's challenges at scale.

In short, you need an integrated environment for design, code, and build artifacts. Helix TeamHub Enterprise, powered by Helix4Git, is an out-of-the-box solution for scaling Git, which we'll use as an example throughout this eBook.

It is possible to make Git work for even the largest of enterprises, scaling to feed giant projects – gigabytes of assets across millions of files – into the DevOps pipeline. This eBook aims to uncover the details of these challenges and introduce five surefire ways to set your organization up for success with Git at scale:



1 Delight All of Your Contributors

2 Examine Your Branching Strategies for CD

3 Scale Git in the Build Process

4 Manage Change Across the Entire DevOps Pipeline

5 Make the Leap from CI to Enterprise CD

1 Delight All of Your Contributors

It all starts with contributors: artists, designers, developers, and many others. Today's multi-disciplinary product development environment involves a lot of input and iteration, both of which lead to explosions of files and collaboration. You need a simple, elegant way to manage multiple projects and multiple teams, not to mention many members who do not possess the same technical skill or even familiarity with Git.

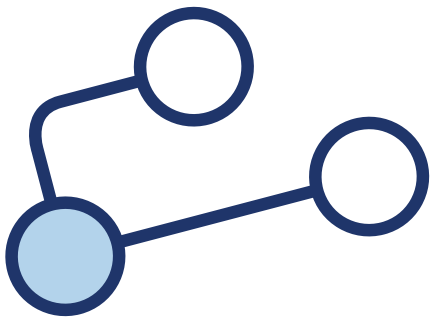
Git was built by a developer for developers, but it can be extremely daunting for others due to its steep technical learning curve. Which is why a true collaboration solution is required to unite all of your contributors in one welcoming place. A collaboration solution should provide:

- A unified web interface that is simple to use.
- Support for projects with multiple repositories and support for systems other than Git.
- A merge-request workflow and reviews to keep developers productive.
- Built-in issue and milestone tracking that everyone can view and understand.
- The appropriate integration power under the hood to feed your DevOps pipeline.

Helix TeamHub provides all of the above – it takes the best of other Git hosting solutions and extends it in significant ways, all while paring down the clutter and keeping the interface simple. And it does this without compromising DevOps needs to mirror/replicate code and scale builds.

In fact, you can create Git repositories, manage cross-repository dependencies and assign repo-level permissions using the intuitive web UI. And, distributed teams around the world can enjoy LAN performance for their daily work while all of their commits are synchronized automatically over slower, WAN links. That's a huge performance boost.

Speaking of performance, developers need quick feedback: the tighter the iteration cycle, the greater the productivity. When developers believe a defect has been fixed, they test it, submit it for code review, and typically wait for a positive response. So the sooner it's released into production, the sooner they can move on to new work.



Delight All of Your Contributors

But developers don't blame Git when its poor performance negatively impacts their productivity. They blame IT, who are then tasked with addressing Git's shortcomings.

This isn't exactly a secret; it's a major topic in online publications and forums. The unhappy fact is that Git cloning and copying can be slow. And this performance hit is only magnified for remote developers in light of how distance, latency, and dropouts loom large.

One way to tackle these problems is to introduce a local proxy/cache at each site. This improves performance by enabling developers to clone from a local server whereas write-back to the master server is accomplished by setting up the remote site properly. This is a do-it-yourself solution, albeit one that is relatively well-documented on internet community sites.

A better solution is to leverage Helix TeamHub Enterprise to increase developer productivity and achieve faster builds. Figure 1 illustrates how it handles concurrent Git checkouts and commits for globally distributed teams.

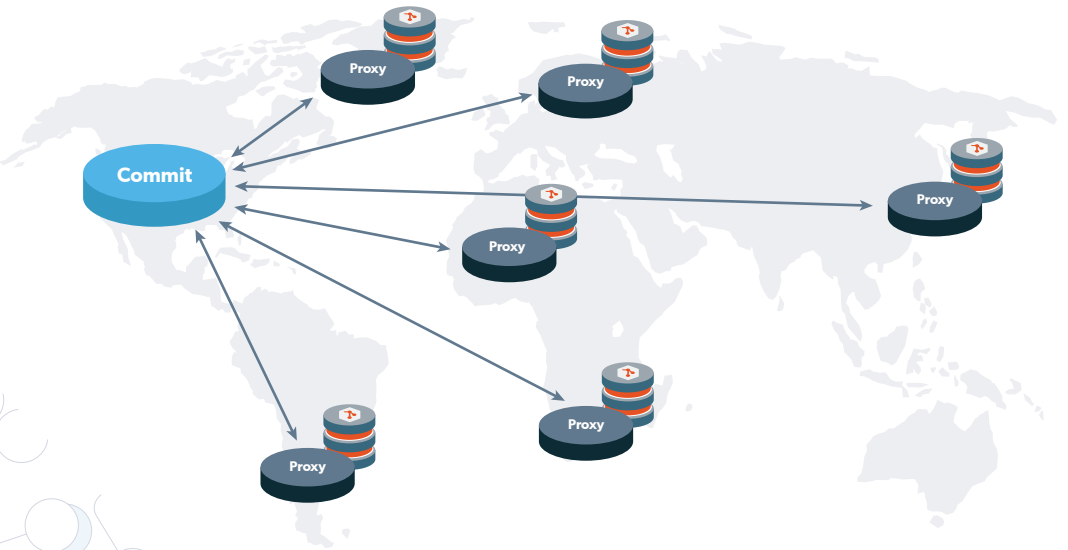


Figure 1 HTH helps global teams handle concurrent checkouts and commits.

2

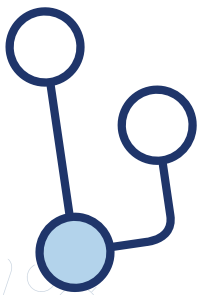
Examine Your Branching Strategies for CD

The correct branching strategy is essential to automating and improving the overall quality of your software. Even though changing branching strategies can be difficult, your present strategy may not be compatible with (or optimized for) automation.

If you regularly spend time figuring out what went wrong when you merged, you are not ready for automation. Smaller, short-lived branches can minimize risk and ensure fewer delays. Let's look at two options.

A SUCCESSFUL GIT BRANCHING MODEL

Many Git teams have standardized on variations of Vincent Driesen's "A Successful Git Branching Model." Its greatest benefit is that it's widely used, and there are numerous variations on the general theme. The downside of this approach, however, is the risk of creating long-lived feature branches, subsequently making everything work together when merged.



KEY COMPONENTS OF DRIESEN'S MODEL

- One centralized Git repo called "origin."
- One production-ready branch called "master."
- One integration branch called "develop."

RECOMMENDED WORKFLOW

- Developers work locally, pulling and pushing from/to "develop".
- Collaborators set up Git remotes so peers can pull changes as needed.
- The continuous delivery pipeline frequently merges changes from "develop" to "master," with as much automation as possible, then releases a new version.

TRUNK-BASED DEVELOPMENT

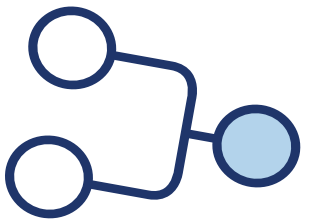
Trunk-based development is an alternate strategy experiencing a surge of interest in the DevOps community. You may benefit from trunk-based development if you are a large organization trying to achieve better quality and faster releases, and/or you operate in an environment where compliance, governance, and security are highly valued.

KEY COMPONENTS OF TRUNK-BASED DEVELOPMENT

- A single, shared branch called "trunk."
- Short-lived feature branches.
- A monorepo strategy.

RECOMMENDED WORKFLOW

- Developers check out very small portions of code, simplifying security and traceability.
- Developers collaborate in the "trunk" and either commit/push (small teams) directly thereto or use pull request workflow (large teams).



Examine Your Branching Strategies for CD

Note: Teams that produce a high commit rate or have many members favor short-lived feature branches for code review and build checking (i.e., CI) before committing work to "trunk." These branches accelerate code reviews, gating what gets added into "trunk."

Small, short-lived branches minimize merge conflict. Whether you use a distributed version control system (DVCS) or a monorepo strategy, your DevOps pipeline goals should revolve around making it easier to introduce new features, fixes, and improve overall code quality. Addressing tooling challenges and adjusting workflow to support these goals go hand in hand.

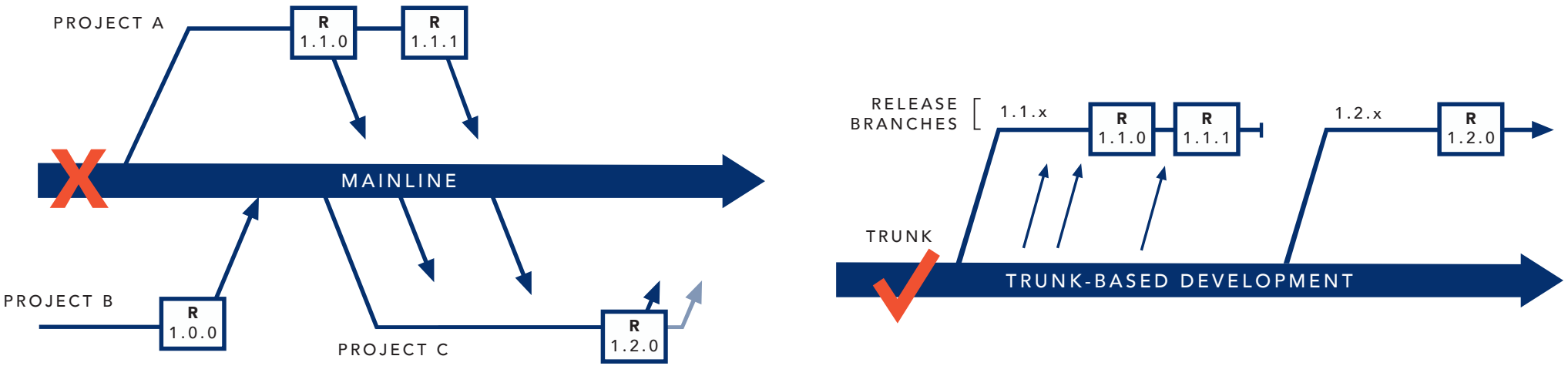


Figure 2 Mainline vs. trunk-based development

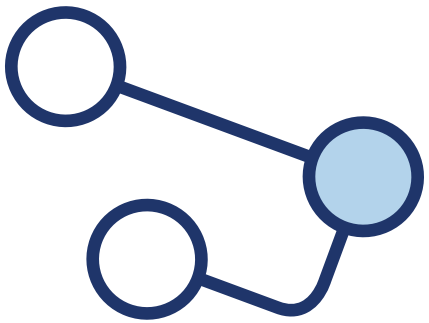
3 Scale Git in the Build Process

Git cloning has emerged as the biggest challenge impacting performance in CI/CD pipelines because Git processes each file individually (i.e. slowly) during such an operation. This happens whether developers are working in the same room or remotely.

Because of these facts, Git itself becomes slow with very large repos beyond roughly 1 GB of content. Splitting a large repo into many small repos can help, but only at the difficulty of bringing everything back together in the DevOps pipeline.

Google recognized this problem with Android and allocated significant resources to create their own repository management tool, Repo. Repo sits on top of Git to handle the very large number of Git repositories associated with Android. However, it only addresses the Android use case, and adds significant complexity at every stage of the pipeline.

Such custom, large-scale tooling works in some cases. But most companies find that it represents a costly, ongoing expenditure that takes focus away from building and shipping revenue-generating products.



TeamHub Enterprise offers a cost-effective, out-of-the-box solution that overcomes the performance challenges associated with many and very large Git repos.

Scale Git in the Build Process

By contrast, Helix TeamHub Enterprise implements a more efficient way of storing Git data than the methodology employed natively by Git. It also uses multiple parallel pipes – with multiple threads pulling less data at the same time to saturate the DevOps pipeline.

TeamHub Enterprise offers a cost-effective, out-of-the-box solution that overcomes the performance challenges associated with many and very large Git repos, while at the same time simplifying CI/CD pipelines by keeping everything together in that all-important "single source of truth" (Figure 3).

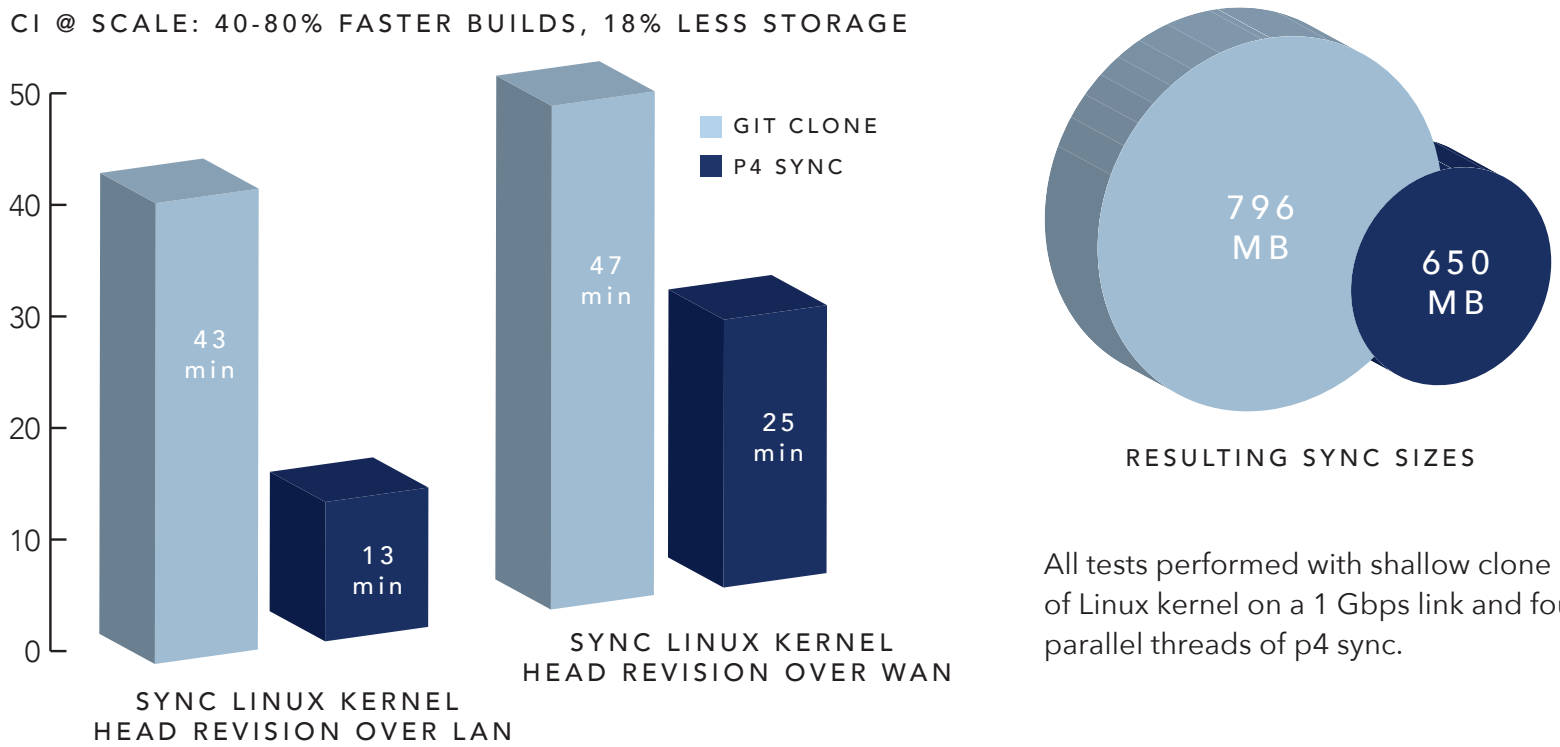


Figure 3 Helix TeamHub Enterprise, powered by Helix-4Git, builds on the proven infrastructure and core strength of the Perforce server, including large scale and performance.

All tests performed with shallow clone of Linux kernel on a 1 Gbps link and four parallel threads of p4 sync.

4

Manage Change Across the Entire Devops Pipeline

In today's market, organizations must continually streamline and automate their DevOps pipelines to remain competitive. This can be particularly challenging for complex projects that include developers, non-developers, and digital content beyond source code – such as graphics, video, audio, and other binary files.

DevOps success relies on merging incoming work to the master quickly, reliably, and often throughout the day. So managing build and release artifacts is just as important as handling source code and other assets. It is crucial to have a "single source of truth" in which everything can be versioned and audited to drive good decisions.

Git's design simply did not anticipate storing and handling large objects. Even the more common tools that attempt to address this gap, such as Git LFS, are neither performant nor user-friendly. They often require designers and other less-technical users to hurdle the steep learning curve of the Git interface. What you need is an actual solution, one that easily accommodates all types of assets and welcomes everyone into the DevOps world.

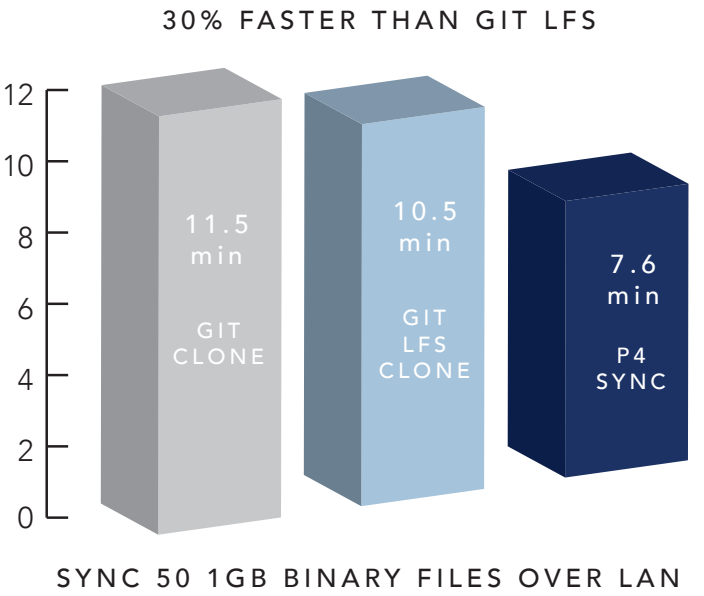


Figure 4 This graphic depicts two repos living together. The Git repository gets the source and the hierarchical repository, with Adobe Photoshop files, movies, and audio formats going in via folders. Compared to Git LFS, Helix4Git is 30% faster.

Of course, you could rely on tools like Git LFS or build your own integration with file-sharing technology like Dropbox. But you would still have disparate workflows and steep learning curves for non-developers. And, in the end, you'd still have to figure out how to move those large assets around the internet, through your own network, and into your DevOps pipeline after unifying it all.

Thankfully, the Helix4Git technology in TeamHub Enterprise allows you to manage large binary files alongside Git source code and dramatically accelerates large file transfers (Figure 4), greatly improving pipeline performance.

5

Make the Leap from CI to Enterprise CD

Many organizations already enjoy the benefits of Continuous Integration (CI), but the largest benefits for the enterprise are yet to be reaped from Continuous Delivery (CD). This kind of automation decreases risk and improves flexibility. Once in place, it also lowers cost and exposes process inefficiencies in real time.

But CD is complicated if you need to support Git teams. It requires teams to automate integrating code (usually at least daily), building, storing binaries back into version control, deploying those binaries to QA, testing, and ultimately pushing to production.

Business stakeholders want the benefits of CD, but also fear that one false move could halt their mission-critical production systems. DevOps professionals know that rewards outweigh risks, especially after taking steps to mitigate those risks. With a solid CI foundation and some hard work, you too can join the ranks of the CD elite.

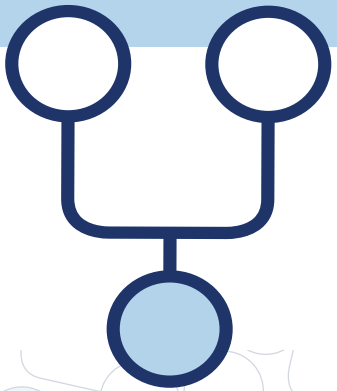
Version control is the bedrock technology for CI/CT/CD because it's the conductor by which the entire CI/CT/CD pipeline is orchestrated. From commits to successful reviews or failed tests, there are so many events that can dictate which actions your systems should take.

The overarching goal of CI/CT/CD is to deliver on the promise of DevOps. Version control drives the delivery of that promise by providing the performance and insight needed to improve software release cycles, software quality, security, and the ability to get rapid feedback on product development.

VERSIONING 101 FOR ENTERPRISE CD:

Version control enables CD. You need a version control robust enough to handle:

- Design assets
- Databases
- Database scripts
- Build tools
- Libraries
- Build artifacts
- And much more

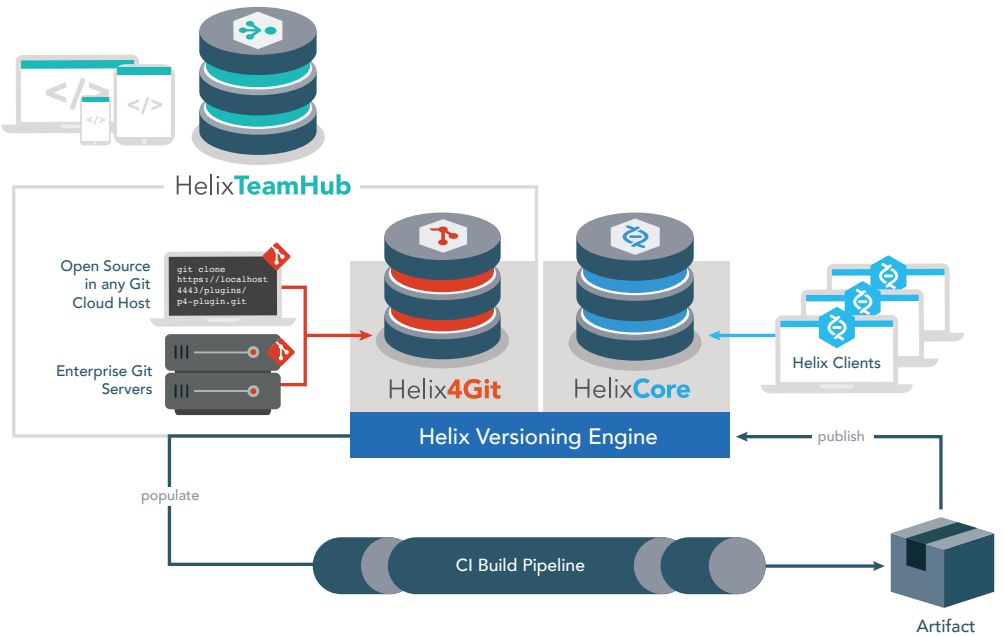


Make the Leap from CI to Enterprise CD

In a perfect world, your DevOps pipeline would be an out-of-the-box, all-in-one solution, but such a tool simply doesn't exist. That's why the Git world boasts such a large community of active contributors, whose creative work allows other users to integrate virtually any tool into their DevOps pipeline.

Similarly, you'll find many open source software (OSS) solutions to improve automation and efficiency. And, of course, add-ons and other systems (Maven, Repo, Artifactory, or others) can help support many large repos, design assets, and build artifacts.

But there is a more streamlined, less patchworked approach. Not only does Helix TeamHub Enterprise store and track all of your assets at each and every stage of the development process, it also has the integration hooks you need to make this an out-of-the-box reality without the heavy lifting. TeamHub Enterprise seamlessly integrates with all of the major CI/CT/CD systems, eliminating the necessity for several other major applications you may be actively maintaining today.



Helix TeamHub Enterprise, powered by Helix4Git, lets you combine your Git source code with all of the other items you need to successfully implement CI. TeamHub Enterprise comes with a number of automation-enabling features, including triggers, ready-made integrations, and a robust API.

Figure 5 Illustrates the importance of version control in the Continuous Delivery pipeline.

Summary

Making Git work in the enterprise is a balancing act between developer satisfaction and overall productivity. But with Perforce you have many options, including tools that were built to withstand petabytes of data worked by tens of thousands of concurrent users in even the most complex and secure DevOps environments.

In fact, Perforce Software has been solving customers' performance and scale challenges from the beginning. Our solutions were built from the ground up to support globally distributed teams that need to move large data sets over long distances and accelerate multi-file, multi-repo product builds.

Helix TeamHub Enterprise makes it easy to scale, automate, and gain visibility into your DevOps pipeline while offering developers the freedom to use their preferred tools. It provides users the ability to support large-scale, Git-based projects with continuous build, integration, and test processes.

By providing a more efficient way to handle Git data, TeamHub Enterprise allows you to close the feedback loop to your developers faster and achieve 40-80% faster builds.

Visit **Perforce.com** to learn more and sign up for a **demo of Helix TeamHub Enterprise**.

United Kingdom
Perforce Software UK Ltd.
West Forest Gate
Wellington Road
Wokingham
Berkshire RG40 2AT
UK
Phone: +44 (0) 1189 771020
uk@perforce.com

Australia
Perforce Software Pty. Ltd.
Level 13, Suite 5
56 Berry Street
North Sydney, NSW 2060
AUSTRALIA
Phone: +61 (0)2 8912-4600
au@perforce.com

Germany
Perforce Software
Zeppelinstrasse 4
Haus 3
85399 Hallbergmoos
GERMANY
Phone: +49 811 998 262 0
dach@perforce.com

Finland
Perforce Software
7th Floor
Jaakonkatu 3 B
Helsinki
00100
Finland

US - Minnesota
Perforce Software Headquarters
400 First Avenue North #200
Minneapolis, MN 55401
USA
Phone: +1 510.864.7400
info@perforce.com

US - California
Perforce Software
2320 Blanding Ave
Alameda, CA 94501
USA
Phone: +1 510.864.7400
info@perforce.com

US - Ohio
Perforce Software Inc.
2320 Blanding Ave
Alameda, CA 94501
USA
Phone: +1 510.864.7400
info@perforce.com

PERFORCE

perforce.com