PERFORCE

MIGRATION GUIDE

IBM ClearCase to Helix Core Migration Guide

Introduction

This document provides information to plan your migration from an existing ClearCase system to Helix Core, the version control system (VCS) from Perforce. Migration projects from ClearCase to Helix Core vary greatly in scale and complexity. Small, simple environments with basic migration requirements are typically migrated in about eight business days. This includes setting up Helix Core, migrating, and training users and administrators. Large, complex ClearCase environments may perform a series of migrations over the course of several months (or more), as teams migrate at times convenient for them.

We discuss preliminary planning, corresponding Helix Core concepts and terminology, and review three history import strategies:

- Starting over
- Detailed history import (DHI)
- Baseline & branch import (BBI)

The Professional Services team at Perforce has experience guiding teams through complex migrations. We can assess your environment and help create your custom migration strategy.

v3, February 2019

2. Preliminary Migration Preparation 1
2.1. Review Existing Branching Strategy 1
2.2. Perforce Directory Standard (PDS) 1
2.3. Release Processes and the PDS 1
2.4. Perforce Streams 2
2.5. Addressing Intellectual Property Concerns 2
2.6. Training 2
2.7. Helix Core Transition Team 2

3. Import Strategies 3
3.1. Tips – Starting Over 3
3.2. Detailed History Import 4
3.2.1. ClearCase Detailed Import History Preparation 5
3.2.4. Detailed History Import – Pros 5
3.2.5. Detailed History Import – Cons 6
3.3. Baseline & Branch Import 6
3.3.1. Baseline & Branch Import – Pros 7
3.3.2. Baseline & Branch Import – Cons 8
3.3.3. Warnings 8

4. Terminology and Concepts 8
4.1. VOBs and Depots 8
4.2. ClearCase Regions 9
4.3. VOB Servers vs. "The Server" 9
4.3.1. Operating System Selection 9
4.4. Registry and License Servers 10
4.5. Release Servers and Installation 10
4.6. View Servers, Protecting Unversioned and Checked Out Files 10
4.7. ClearCase MultiSite vs. Perforce Federated Architecture 11
4.8. Replacing ClearCase Views with Perforce Workspaces 11
4.9. Rethink Label Strategies 12
4.10. Unified Change Management (UCM) 12
4.11. Migration Technical Details 12
4.11.1. Evil Twins 12
4.11.2. Symlinks on Windows 13
4.11.3. File Type Mappings and Limitations 14

5. Conclusion 14

2. Preliminary Migration Preparation

2.1. REVIEW EXISTING BRANCHING STRATEGY

Early in your migration planning, determine whether your current branching strategy used in ClearCase is appropriate to use going forward. If not, adjust your strategy as needed.

Creating an initial branching strategy is a best practice when getting started in Helix Core. Perforce Streams can model your branching strategy and indicate the intended flow of change.

2.2. PERFORCE DIRECTORY STANDARD (PDS)

With Helix Core, the directory structure and branching strategy are related. A well-designed directory structure is critical because it:

- Conveys branching patterns for everything in Helix Core.
- Maps change propagation paths for the various flows of change. For example, the life of a bug discovered in maintenance, or the life of a new feature.
- Conveys the stage in the life cycle of any particular piece of code: experimental, development, or released.

It's a best practice to create a Perforce Directory Standard (PDS) to establish the directory structure and corresponding branching strategy in Helix Core.

2.3. RELEASE PROCESSES AND THE PDS

The directory structure in Helix Core can be thought of in "low" and "high" levels. Low levels represent your software products, and can vary for each software product. High levels of a Perforce directory structure convey branching structure, project management, and software lifecycle information. A well-designed, high-level directory structure is intuitive for developers and lends itself well to project management metrics, policy enforcement by branch type, and various kinds of automation.

Migrating to Helix Core typically involves defining a Perforce Directory Standard (PDS) for each product that is imported into Helix Core. In some cases, it can be used across the entire organization. A PDS encourages consistency in release processes for various software products. It can be as flexible as needed to account for the different release processes and branching patterns for various software products in Helix Core.

For example, one software product might be licensed, which might have a release process that defines how to maintain old releases and deliver patches. A web-based software product, in contrast, might not require maintenance of old releases, but must support rapid updates. Still another product could have a set of generic components that are delivered to customers then heavily customized, perhaps by your own organization.

Release processes for different software products may also vary due to the number of contributors and the structure of QA processes. Software products can follow the same release process, even though they might have very different release schedules.

Low levels of the directory structure are left untouched by the migration. This minimizes the difficulty of performing the migration and the impact of the migration to your environment (e.g. build scripts, release processes and tools, etc.).

2.4. PERFORCE STREAMS

Perforce Streams models branches at a higher level. A stream defines multiple things:

- What related files are included in a set.
- Where those files came from (the parent stream).
- How change flows to other streams.
- How parts of a stream are treated in the workspace.

Streams is an attractive framework for new projects in Helix Core. The tools and workflow improvements simplify work for end users based on industry best practices. It also makes project and release management easier.

The detailed history import (DHI) tool used by Helix Core currently does not support ClearCase data being imported directly into Streams. However, ClearCase data can be imported into "classic" Helix Core branches and then migrated to Streams for future work. Alternatively, the baseline & branch import (BBI) method can be used to import ClearCase data directly into Streams.

2.5. ADDRESSING INTELLECTUAL PROPERTY CONCERNS

Maintaining IP provenance (i.e., knowing where your source code came from and knowing what legal rights you have to it) can be made a priority during the migration. When looking at the migration process, your goal should be to ensure that IP provenance is not negatively impacted. Your migration processes should provide a clear audit trail so that all imported files can be traced back to the original ClearCase repository.

VCS systems inherently store valuable intellectual property. If sensitive information is being migrated, both the migration process and the resulting Helix Core environment should ensure that access is controlled to the same degree as it was in ClearCase.

Migrations also provide an opportunity to review access control policies. This process can help expose particularly weak access controls, and highlight where controls have gone too far. In some cases, ensuring strong IP protections requires extra effort. It is important to consider if strong access controls really benefit your organization.

The powerful and flexible access control capabilities available in Helix Core provide a straightforward means of guarding IP with relative ease.

2.6. TRAINING

Training for Helix Core users and administrators is essential to help a migration go smoothly. We find it most effective to train the bulk of users a few days to a few weeks prior to the migration to Helix Core.

Perforce provides a variety of training options including in-person and online instructor-led training. It is most effective to provide instructor-led training to a core set of administrators and key users (e.g., "train the trainers").

2.7. HELIX CORE TRANSITION TEAM

We recommend establishing a transition team during your migration process. This core group may include application administrators, system administrators, and other influential users. You

might consider engaging Perforce Consulting to be a part of your transition team.

The transition team defines how Helix Core will be used in your organization:

- How it will tie into your various processes and workflows?
- How should it will be integrated with other systems?

For larger and more complex migrations, training for this team should occur early in the planning process. This allows best practices established by the team to evolve, be documented, and be communicated to the larger user community.

3. Import Strategies

There are three supported approaches for importing files:

- Starting over (Tips)
- Detailed history import (DHI)
- Baseline & branch import (BBI)

The following is an overview of the strengths and limitations of each import strategy.

3.1. STARTING OVER STRATEGY - TIPS

Starting over really isn't really a conversion strategy. This approach uses the "tips" — the / main/LATEST file versions from ClearCase – and simply adds them to Helix Core without any history.

Based on your organization's Perforce Directory Standard, a high-level directory is identified in which the files will be stored. For example: //Janus/main/src

In this example, Janus is a product name. Main indicates files in the main stream of development. Src is the root of the low-level directory tree. The low-level directory tree is copied verbatim into Helix Core.

The "Tips approach" is sometimes appropriate for specific parts of a project. For example:

- Documentation VOBs
- VOBs for shelved but not terminated projects.

This approach is usually not appropriate for source code, except for prototype and demo code.

Even in simple Tips migrations, care must be taken to ensure that file types are mapped correctly. Text, binary, and Unicode files should be reviewed. Also file type modifiers should be applied when migrating, such as '+x' for executables and '+F' for compressed binary formats like *.gz or *.mpg, etc.

Starting over offers some benefits. First, it can be easy. You define target directories in Helix Core, and then add the files. Because there is less to migrate, it is also faster.

But for organizations that need to remain in compliance, this approach does not move historical information into Helix Core. If multiple branches are imported, Helix Core won't be able to simplify first-time merges between them. In order to easily perform these merges, Helix Core needs to understand the historical relationship between the branches.

3.2. DETAILED HISTORY IMPORT (DHI)

Detailed history import (DHI) is the logical extreme migration case. The goal with this approach is to capture as much detailed branch/merge information and migrate it into Helix Core. This ensures that comprehensive historical research can be done in the new system, without the benefit of the old.

Perforce has a sophisticated tool and process that enables conversion of very detailed historical data from ClearCase to Helix Core. The tool has been successfully used for performing detailed history conversions from fairly large ClearCase data sets (about 100 GB) to Helix Core. Due to licensing restrictions and operational complexity, the DHI conversion tool is only available through Perforce Consulting services.

Detailed history migrations from ClearCase can be selective. You can select only a subset of VOBs to be imported. Within each imported VOB, a subset of all available files and labels are typically targeted for import.

Conversion includes file contents for each revision. Details include the metadata associated with each check-in, file renames, and detailed branching and merging history. ClearCase's representation of branching activity is translated into Helix Core as "integration records."

The import process groups ClearCase "check-ins" into Helix Core "changelists." For example, if a given user checked in a set of files at the same time (+/- a few minutes) with the same check-in comment, those would be grouped together as a single Helix Core changelist. UCM metadata, if available, also factors into changelist grouping.

Our migration process starts by scanning repositories for various issues that need to be resolved in ClearCase prior to migration. Examples include:

- "Evil twin" files and directories.
- File types such as "block special devices" that are supported only in ClearCase.
- File types (such as hard links) that can be imported, but require special emulation in Helix Core.
- Architectural and conceptual differences between ClearCase and Helix Core. For example, mapping ClearCase labels into Helix Core proves rather difficult, because typical usage of labels varies between the systems.
- Certain rare "circular merge" activities that are not easily translated into Helix Core.
- ClearCase repositories that have mild data corruption that might not be visible to normal users, but would be exposed when a detailed conversion history tool runs specific commands.

DHI has a more involved undertaking with ClearCase than with other VCS systems. Concepts from UCM and RUP that depend on ClearCase attributes and triggers are not handled by import tools. They require manual importing.

Depending on the amount of data being migrated and the speed of ClearCase in your environment, a complex migration strategy using imports running in parallel may be required. Extracting all information from ClearCase – with years of

history – might take a full week to perform the mechanical extraction/import.

3.2.1. CLEARCASE DHI PREPARATION

If you plan to engage Perforce to help you plan a detailed history migration from ClearCase, there are a few things to be aware of early in your planning process.

You'll first need to establish a replica of your ClearCase environment, including all VOBs targeted for import. This instance should be provisioned on hardware separate from your production environment. This allows dry runs to be completed without taxing a production server.

The migration will also require a powerful, dedicated migration server. This can be the new Helix Core server, but it does not have to be. It must be a Linux server, even if the production and replica servers are Windows. A fast network connection between the ClearCase replica machine, the migration server, and the Helix Core server (which can be the same machine) is essential.

Migration is a very resource-intensive process, and can potentially be very demanding in terms of disk space and RAM resources. It is more demanding than actual Helix Core server operations, as the equivalent of years of work done in ClearCase is compressed into hours or days to get it into Helix Core.

Detailed migration often involves custom scripting due to differences in ClearCase usage, oddities in a particular data set, and custom requirements. This is why it is important to review all your requirements prior to migration.

3.2.2. DETAILED HISTORY IMPORT – PROS

Using a DHI strategy gives you the ability to view file history using powerful visualization tools from Helix Core, such as Time Lapse View. This can shed new light on the evolution of your source code, and give you a better understanding of changes over time.

There is also an increased benefit for systems integrated with version control. For example, the meaning of the linkage between a set of files originally modified in ClearCase, and an issue from your issue tracking system, can be maintained. Plus with code review tools like Helix Swarm, you can continue to provide greater context to future changes.

Unlike Helix Core, ClearCase does not have a way to validate the integrity of versioned file contents using checksums. Corruption of file contents – e.g. due to disk failures – can go undetected[1]. Once historical data is in Helix Core, it will benefit from checksum verification, which will improve IP provenance. This allows your teams to access a file's history that before might have been impossible.

After the migration, comprehensive historical research and "merge forensics" can be done in Helix Core without the need for going back into ClearCase. Although, if possible, it is recommended you keep ClearCase as a backup with one user license for a few years.

3.2.3. DETAILED HISTORY IMPORT – CONS

Detailed import tools have a variety of technical limitations. Some of these limitations are due to differences in the way ClearCase and Helix

[1] *ClearCase does have a 'checkvob' utility that can detect and fix some forms of metadata corruption. However, this utility does not detect data container corruption, and thus the contents of versioned files cannot be audited.*

Core work. Others are due to the potential complexity of ClearCase environments, including unusual patterns (or even corruption) in the data. So-called "evil twin" elements and certain circular branching patterns created by misconfigured config specs can be difficult to follow.

Existing detailed history import tools may require development to work on your data prior to migrating. The likelihood of this scenario depends on your data. It is typically necessary if a full-context migration is attempted. Added complexity translates into potential schedule and budget risks for your migration project.

3.3. BASELINE & BRANCH IMPORT

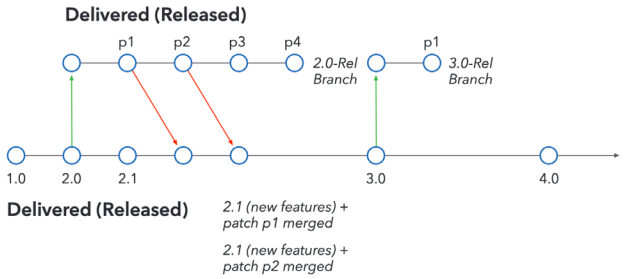
The baseline & branch import (BBI) strategy provides a lightweight migration alternative. It is more sophisticated than the simple Tips approach, yet without the technical complexity involved in detailed history import (DHI).

The BBI process is generic and can be done from any other version control system to Helix Core. Customers have used it to migrate from a variety systems – IBM ClearCase®, Borland StarTeam®, Merant PVCS®, Subversion, Mercurial, CVS, Microsoft Visual Source Safe, and AccuRev. It has even been used to migrate from a set of network drives with directories named to indicate releases.

With the BBI approach, only certain points in your history are imported. The branch diagram shows the baselines – snapshots of a directory structure at a point in time – and major branching operations.

Sample Baseline & Branch Diagram

Figure 1: Sample Baseline & Branch Diagram



The baselines (blue dots) indicate what "interesting versions" are to be imported. The arrows indicate major branching operations that affect an entire branch. In this scenario, a 2.0-Rel branch has been created along with four patches on that branch. When migrating to Helix Core, we can see only two of those four patches have been merged back to MAIN. The BBI process:

- Imports all the baselines.
- Records the fact that two merges were completed with updates added to MAIN.
- Tracks the two unmerged patches remaining on the release branch.

Once all this information is available in Helix Core, it can be used to complete new merges. Importing the branching operations allows Helix Core to select common ancestors for merge work. After the migration, your teams can pick up right where they left off with branching activities.

The BBI process imports branching operations at a high level, capturing the sum of merge operations. For example, in Figure 1, the arrow representing the merge of p2 back to MAIN would likely have occurred as a series of merges

carried out by several developers. The individual file merges are not tracked, but the sum of the results of the merge – file adds, edits, and deletes – are tracked. The imported baseline represents a point in time when the merge of p2 is considered complete.

The goal with BBI is to bring over just enough branching history to answer key questions. For example:

- What did release 2.0 look like?
- Where was this file branched from?
- What files do I need in my workspace to start maintenance work on release 2.3?

Because the BBI approach preserves file contents at key points, the cutover to Helix Core can happen at any point in the release cycle.

After conversion, Helix Core would show history of your software product in its Revision Graph tool. This view will look like the product was developed in Helix Core from the beginning. Although detailed data is lost, you will know what the state of your product looked like at each release. But the hundreds of check-ins between those baselines are discarded, as are user-ids, dates, times, and check-in comments.

Accurate diagrams are essential for planning a BBI migration. Ideally, release engineers should draw a branch history picture for each software product to be imported. This information can also be manually found in ClearCase. Once the diagram is drawn and vetted, it is translated into a set of Perforce commands that recreate the baselines in Helix Core.

The first baseline will appear as an initial addition of the entire product directory tree. Subsequent baselines result in Perforce changelists that show only certain changes like files added, deleted, or modified. Branching operations are translated into Helix Core equivalents. Merges done in ClearCase are recorded in Helix Core as the results of the merges.

If your organization will still need access to detailed historical research, ClearCase can be kept running with a single license. It is a good idea to keep this ClearCase license around at least for a year or two after a BBI migration.

3.3.1. BASELINE & BRANCH IMPORT – PROS

When implementing a BBI strategy, ideally you would want the flexibility to do a multisystem migration for different teams. Each team could potentially migrate into Helix Core on their own schedule and without impacting others. Because the BBI approach works against a live, running Helix Core server (rather than generating separate server instances like some detailed history import tools) the project planning for each team does not require coordination.

Once your "interesting history" is available in Helix Core, you can use powerful file and directory diff tools – Revision Graph and Time Lapse View – to view your old files in a new light. Unlike the detailed imports, you won't be able to tell exactly who changed what, when, and why. But you can tell how the software product evolved from baseline to baseline.

The BBI process is fairly straightforward, and has little risk of technical snags. Compared to detailed options, this approach makes migration partic-

ularly easy. All the historical information can be loaded into Helix Core prior to migrating teams. Then, on the day of cutover, only the baselines representing the latest state of development on active branches needs to be brought in.

BBI also runs very quickly. Because of this, dry runs can easily be done to test changes that may need to be part of the migration, such as updates to build scripts or makefiles. And the amount of metadata resulting from BBI is negligible and does not unduly impact performance or initial capacity planning.

Migrating to a new VCS tool gives you the opportunity to normalize past history into a new PDS. This allows you to create consistency across activities, such as the creation of branches. In cases where branching strategies evolved over time with ClearCase, you can simplify historical research of the imported baselines. With the BBI approach, common concepts such as "software product X went to production" can be indicated the same way for each of the imported software products.

3.3.2. BASELINE & BRANCH IMPORT - CONS

In cases where files were renamed or directory structures were reorganized between releases, the historical connection between files' names can be difficult to capture. For example, file hello.c in v1.0 of your software product was renamed greetings.c in v1.1. The fact that greetings.c used to be hello.c requires analysis of your data to detect.

Both ClearCase and Helix Core track renames, but each in their own way. But in BBI migrations, that historical linkage of the renaming is sometimes forgone. Renaming can be easily captured – as opposed to showing a delete of one file and the adding another – without the connection that the two are related.

ClearCase allows versioning of some uncommon, low-level file types on some platforms. These block special devices and character special devices are not supported in Helix Core. Such files will not be imported with BBI or any migration strategy. Symbolic links can be imported, however.

3.3.3. WARNINGS WITH BBI STRATEGY

If version control best practices were not followed with ClearCase, reproducing the baselines may be difficult[2]. For example, if branches were made from /main/LATEST rather than from a label on MAIN, getting a config spec to represent the baseline from which a branch was created may involve some guesswork. You may need to use /main/LATEST with a '–time' clause and select a "bestguess time" to represent the point on MAIN at which a branch was created.

4. Terminology and Concepts

In this section, we will review the Helix Core equivalents of basic ClearCase concepts. Consider these when planning your migration.

4.1. VOBS AND DEPOTS

A Helix Core depot is roughly equivalent to a ClearCase VOB (Versioned Object Base). VOBs and depots both display as top-level directories to users, and store a set of files. At least one VOB or depot must exist before any file can be versioned.

A VOB is a container for versioned file contents and metadata related to those versioned files. A

[2] *Unfortunately, this is a common scenario with ClearCase.*

Helix Core depot contains only the contents of versioned files. All metadata is stored in a central database on the Helix Core server.

When mapping VOBs to depots, consider the following:

- Unlike files in ClearCase VOBs, files can be branched across depots in Helix Core. Depots are more transparent (barring any special access controls).
- In Helix Core, you can manage all digital assets – including binary files. Most customers do not manage binaries in ClearCase due to performance considerations. When mapping, you can create a depot for source code and other various components of software products. For example, a depot might be assigned fore each, e.g. //gizmo, //gizmo-build, and //gizmo-release.
- Depots can be created in seconds, but can last a lifetime. Choose their names wisely. Names should be kept short to allow them to be referenced easier. For example: //Engineering is OK, but //Eng is better.
- Path names are primary keys in many Helix Core databases. Character limits are platform dependent and no less than 1024 characters.

4.2. CLEARCASE REGIONS

In ClearCase, network registry regions can be employed to segregate VOBs. These regions restrict a user to see only a subset of all VOBs.

To achieve similar segregation in Helix Core, the P4 Protections Table can be used. Users who were in different regions in ClearCase would be assigned to different user groups in Helix Core. Access to different depots would then be managed at the group level.

4.3. VOB SERVERS VS. HELIX CORE SERVER

In ClearCase environments, there may be multiple VOB server processes potentially distributed across multiple VOB server machines. With Helix Core, a single server may be adequate for any given installation[3]. The process easily runs on a single machine and is frugal with system resources when compared to ClearCase. One Helix Core server can scale to support extremely large environments (e.g. 10,000+ users) using enterprise-grade server machines.

One of the first steps in any migration is to setup Helix Core hardware in a data center. See the General Performance Recommendations for information useful in capacity planning for your primary server.

It is also common to allocate two or three identical server machines to Helix Core to achieve High Availability and Disaster Recovery (HA/DR) goals. A typical configuration has two servers (a primary and a hot spare) in a primary data center, and a third server (warm spare) in another data center located far from the primary data center.

4.3.1. OPERATING SYSTEM SELECTION

Just as with ClearCase, the primary factor in selecting an operating platform for Helix Core is the platform that IT is most comfortable supporting.

[3] *Deploying multiple Helix Core server instances within an enterprise is common. For purposes of this paper we consider only the single-server-per-enterprise approach, as that best suits most ClearCase migration scenarios.*

In mixed Windows/Linux environments, Linux platforms are almost invariably selected for ClearCase. This is primarily due to case sensitivity reasons and ClearCase's reliance on a monodirectional filesystem. VOB data is then accessible to clients on both Linux and Windows.

With Helix Core, only a Transmission Control Protocol (TCP) connection is needed between clients and servers. Further, the case sensitivity behavior of Helix Core can be configured independently of the platform. Thus, with Helix Core, the server can be configured on Windows or Linux in multi-platform environments.

4.4. REGISTRY AND LICENSE SERVERS

Helix Core does not require license or registry server processes or additional hardware. A simple 0.5 KB license file on the Helix Core server machine drives the license mechanism.

4.5. RELEASE SERVERS AND INSTALLATION

With ClearCase, it is important to carefully manage server and client versions. Ensuring users run client software that is compatible with the current version of the server is important to ClearCase. To ensure consistency, some ClearCase installations deploy a separate release server. This defined network resource allows users to download correct versions of client software.

With Helix Core, all server and client components install in minutes over the web. More importantly, Helix Core clients and the server have a very loose forward and backward compatibility relationship. Users can generally run client versions that are older or newer than the server. Client programs simply hide or disable those features that require newer versions of the server, and new server versions rarely require client upgrades.

Helix Core supports a centrally configured, automatically deployed client for large Windows sites (see Automated Deployment of Perforce). Such sites can be used to ensure that users download consistent, trusted versions of software that are supported by IT and/or release engineers. The ease of installation and enhanced compatibility makes maintaining clients less of an administrative priority than in ClearCase environments.

4.6. VIEW SERVERS, PROTECTING UNVERSIONED AND CHECKED OUT FILES

Helix Core stores all metadata in databases located on the central server[4]. There is no equivalent of a ClearCase view server process. Because of this, there is no need for administrators to allocate and configure view server machines in Helix Core. Users also do not need to start or stop view server processes.

When dynamic views are used, view server machines contain the contents of checked-out and unversioned private files. Some organizations back up view storage areas regularly to protect against loss. If protecting checked-out and unversioned files is a priority, you will need to address this when migrating to Helix Core. In Helix Core, unversioned and checked-out files are not available to the server, and are not backed up.

Some organizations devise infrastructure to help protect unversioned and checked-out workspace files in Helix Core. They might require users to store workspaces on network drives that are backed up. More often, you can addresses these processes and users during training, and encour-

[4] *Some GUI programs temporarily cache metadata in running processes, but such information is not persisted.*

age them to avoid keeping files checked out for too long.

4.7. CLEARCASE MULTISITE VS. PERFORCE FEDERATED ARCHITECTURE

If your ClearCase environment relies on MultiSite, Perforce Federated Architecture provides flexible and superior support for remote development.

Helix Core proxy servers are both simple and effective. Helix Core proxies cache the contents of versioned files at remote sites, greatly reducing the dependency on WAN networks. Proxies do not cache any metadata. This ensures one single source truth and eliminates the need for specialized and administration-intensive concepts like branch mastership, scheduling batch replication, etc. Once hardware is available, Helix Core proxy servers can be setup in minutes and require virtually no maintenance.

Helix Core replicas are more capable than proxy servers and have a small administrative footprint. One popular replica configuration could provide read-only access to all data (file content and metadata). This improves performance for remote teams and automated processes like CI/CD. Build performance is enhanced because all read-only data is serviced locally. Data consistency is still maintained because there is still one canonical representation of important data on the central server.

Replicated VOB servers generally run on server machines equivalent to the primary server. It requires the support from a similar tiered data center. Due to the significant investment in licenses, hardware, and administrative overhead, MultiSite installations are used only in cases where major development centers exist. There are rarely implemented where many small teams are spread out.

By contrast, Helix Core proxy servers are lightweight programs that can run on desktop-grade hardware, even in enterprise environments. Proxy servers are so lightweight in terms of hardware resource demands that they can be deployed anywhere that even a few developers gather. In some cases, individual users deploy a Helix Core proxy instance in small office without IT support.

Helix Core replicas require more powerful hardware than a proxy server, but have a small administrative footprint. They can be useful for small or medium sized teams, as well as larger sites. There is no additional cost or license issues to deal with when deploying a Helix Core proxy or replica servers.

4.8. REPLACING CLEARCASE VIEWS WITH PERFORCE WORKSPACES

The term *workspace* is familiar to both ClearCase and Helix Core users. In both systems, it refers to what developers use to manage files under version control on their local machines.

With ClearCase, it is typical for a developer to maintain several workspaces, or views. Developers working on multiple branches typically use a different view for each activity. They work in one view at a time. For example, a developer might maintain a liz_user_main_dev view with a config spec selecting /main/LATEST versions, and a separate liz_user_rel_2.3 view selecting /main/REL2.3/LATEST[5] versions.

[5] *This is an oversimplification. A typical config spec consists of several lines or more.*

A Perforce client spec – a form controlling the definition of a workspace – determines the subset of Helix Core files visible in a workspace. To users, branches in Helix Core display as fully populated directory trees. There will typically be a directory on the server named MAIN, and another directory named for a specific release. For example, REL2.3. A developer might have a liz_user_dev workspace, which could include both MAIN and REL2.3 directories. Liz could easily work in both activities at the same time.

In Helix Core, only user files are stored on the local disk. All metadata – including information about the name and contents of a user's workspace – is maintained on the Helix Core server. If your teams are using Perforce Streams, they have the capability to specify which modules are actively in use in a branch, and import dependencies from other projects.

4.9. RETHINK LABEL STRATEGIES

Both ClearCase and Helix Core provide labels, which can be used to identify the versions of files that constitute a baseline. For many ClearCase users, the use of labels is mandatory. But applying labels can be time-consuming. This task could account for 30% or more of the time associated with official builds.

In Helix Core, labels are just one way of reproducing baselines. Labels certainly do the job of identifying a baseline, but other approaches are available. You can accomplish the same thing without the taxing build process. Alternatives to labeling take advantage of Perforce changelists. Each Perforce check-in generates a unique changelist number that reflects the state of the entire repository at a point in time. Any given changelist, even though it affects only a small subset of the files, can be used to describe the state of every file in the depot.

Branches in Helix Core are represented as directories, making it easy to use a combination of branches and changelist numbers to represent a baseline. Alternately, labels can reference changelist numbers limited to an identified scope in Helix Core, where the scope is typically the directory representing a particular branch.

4.10. UNIFIED CHANGE MANAGEMENT (UCM)

UCM adds a layer of process to base ClearCase. We strongly recommend reviewing our Streams Adoption Guide.

Perforce Streams provides a flexible workflow and guidance for release management. When used in conjunction with other ALM tools, like Helix ALM, it can replace ClearCase UCM.

4.11. MIGRATION TECHNICAL DETAILS

Perforce is *not* ClearCase. ClearCase and Helix Core think very differently. They have very different internal representations and modeling of parallel development, branching, and merging. The net result to the user is that both provide good model of what you need to do to achieve parallel development – but one should be aware of the differences.

4.11.1. EVIL TWINS

An "evil twin" is a scenario where two elements with the same name appear in different branches. For example, say you have a MAIN, DEV_A and DEV_B branches, with each of the DEV branches parented directly from MAIN. In DEV_A branch,

a developer runs **ct mkelem** to create a new file element, foo.c. Independently in DEV_B branch, a developer runs the same command to create a new file element, foo.c.

Then another developer runs **ct findmerge** to make files that originated on DEV_A appear on MAIN. Later, someone does another **ct findmerge** intending to merge changes from MAIN to DEV_B, including the new foo.c merged to MAIN earlier from DEV_A.

Which is the real "foo.c" in DEV_B? The one that originated in DEV_A, or the one that originated in DEV_B? To ClearCase, it is unclear. One is identified as the correct file, and the other is dubbed the evil twin. Although one would expect them to be branch-relations of the same element, in ClearCase they are not related. They are identified as completely independent elements, referenced in its database by different OID (object identifiers).

If the **findmerge** command completes successfully, you would have two foo.cs in the MAIN branch. But only one foo.c would display in the directory. It is not obvious to users which file is being referenced. ClearCase doesn't provide any sort of warning. Situations are even worse when there turn out to be "evil twin" directory elements.

"Evil twin" directory elements are one of the more insidious complexities of ClearCase. ClearCase admins that are aware of this potentially confusing scenario sometimes put in "evil twin detection" and "evil twin prevention" triggers.

From the perspective of migrating data from ClearCase, this "evil twin" history is murky and not something to bring forward into Helix Core. The process for dealing with evil twins in Helix Core is to detect instances of them, manually select the "correct" element from the pair, and use **ct rmelem** to eliminate the evil twin.

In Helix Core, it is possible for a similar scenario to occur. A file with the same name could be created independently in two branches. However, unlike in ClearCase, Helix Core detects and encourages you to resolve the situation the first time they are merged together. The histories of the two files can be combined in Helix Core, and their revision histories united. You are not required to kill off an evil twin. Therefore you do not lose any history. Instead you can simply unite the trees.

4.11.2. SYMLINKS ON WINDOWS

Using Multi-Version File System (MVFS) with ClearCase enables support for symlinks on Windows in dynamic views. Helix Core has no equivalent of a custom filesystem, and does not support symlinks on non-supported platforms. Symlinks are supported on Windows Vista, Windows 7, and Windows Server 2008, but not on earlier versions of Windows. If there is a reliance on using symlinks on earlier versions of Windows, this must be accounted for during migration planning.

Helix Core does allow symlinks to be versioned. When a file of type 'symlink' is brought into a Helix Core workspace on a Windows machine without symlink support, it displays as a text file. The contents are the target path of the symlink. For example, say in a Linux workspace you did 'ln

–s hello.hpp hello.h'. The file hello.h would be a symlink pointing to hellol.hpp.

In Helix Core, if you sync the hello.h symlink to a Windows workspace without symlink support, you get a file with the contents being "hello.hpp", the path to the target of the symlink. You would not get the content of the pointed-to file, as would occur in a ClearCase snapshot view.

4.11.3. FILE TYPE MAPPINGS AND LIMITATIONS

When migrating from ClearCase to Helix Core, the 'typemap', which defines file types based on Helix Core pathname heuristics, will help ensure that files are added with the correct file type mapping. This is especially important for Unicode files.

ClearCase allows versioning of some file special filesystem objects, such as block special devices, character special devices and hard links. These have no equivalent in Helix Core. If such objects are versioned in ClearCase, you will need to account for that in your migration planning.

5. Conclusion

After reviewing our guide, use our ClearCase to Helix Core Migration Planning Checklist to start preparing for your move.

Not sure where to start?

The Professional Services team at Perforce has extensive experience migrating complex ClearCare environments to Helix Core. They can ensure your migration goes smoothly. For more information, visit https://www.perforce.com/support/consulting or email consulting@perforce.com.

About Perforce

Perforce powers innovation at unrivaled scale. With a portfolio of scalable DevOps solutions, we help modern enterprises overcome complex product development challenges by improving productivity, visibility, and security throughout the product lifecycle. Our portfolio includes solutions for Agile planning & ALM, API management, automated mobile and web testing, embeddable analytics, open source support, repository management, static & dynamic code analysis, version control, and more. With over 9,000 customers, Perforce is trusted by the world's leading brands, including NVIDIA, Pixar, Scania, Ubisoft, and VMware. For more information, visit www.perforce.com