PERFORCE

Perforce Software, Inc.

510.864.7400

consulting@perforce.com

www.perforce.com

# Legacy SCM Migration Strategies

Document ID: LegacySCM-MigrationStrategies.docx v1.4 (May 10, 2019)

# Table of Contents

1 Introduction ..... 1

2 Migration Preparation ..... 1

2.1 Review Existing Branching Strategy ..... 1

2.2 Perforce Directory Standard (PDS) for Helix Core ..... 1

2.3 Release Processes and the PDS ..... 2

2.4 Addressing Intellectual Property Concerns ..... 2

2.5 Training ..... 3

2.6 Perforce Transition Team ..... 3

3 Import Strategies ..... 3

3.1 Tips – Starting Over ..... 4

3.1.1 Starting Over - Pros: ..... 4

3.1.2 Starting Over - Cons: ..... 4

3.2 Detailed History Import (DHI) ..... 4

3.2.1 Pros: ..... 5

3.2.2 Cons: ..... 5

3.3 Baseline & Branch Import (BBI) ..... 6

3.3.1 Pros: ..... 8

3.3.2 Cons: ..... 9

Copyright © 2008-2019, Perforce Software. All rights reserved.

# 1 Introduction

This document provides information for planning a migration from a legacy SCM system to Perforce Helix Core.

We discuss preliminary planning topics and review a variety of history import strategies. In particular, a lightweight migration strategy known as the *baseline & branch import strategy* (BBI) is explored in detail. The BBI strategy provides an alternative to detailed history import strategies, which can be complex due to architectural and conceptual differences between Helix Core and your legacy SCM system.

Helix Core migration projects vary greatly in scale and complexity. Small, simple environments with basic migration requirements are typically migrated in about eight business days, including Helix Core server setup in accordance with best practices, branching strategy development, legacy SCM data migration, and training for users and administrators. Large, complex environments may perform a series of migrations taking several man-months of effort that occur over the course of a year or more, as teams migrate at times convenient for them.

This document is not intended to be a replacement for an actual assessment of your environment. An assessment would produce a tailored migration strategy, with focus on those factors most relevant to your environment.

# 2 Migration Preparation

## *2.1 Review Existing Branching Strategy*

Early in migration planning, determine whether the current branching strategy used in your legacy system, if any, is appropriate to use going forward with Helix Core. If not, adjust the strategy as needed first. Creating an initial branching strategy is a best practice when getting started in Helix Core in any case. This is especially the case in a migration scenario if any history is to be preserved.

## *2.2 Perforce Directory Standard (PDS) for Helix Core*

With Perforce Helix Core's Inter-File™ branching mechanism, the directory structure and branching strategy are related. A well-designed directory structure in Helix Core is critical, because it:

- helps convey branching patterns,
- helps intuitively map change propagation paths for the various flows of change (e.g. the life of a bug discovered in maintenance, or the life of a new feature), and
- conveys the stage in the life cycle of any particular piece of code (experimental, development, tested, formally released).

Copyright © 2008-2019, Perforce Software. All rights reserved.

It is a best practice to establish a directory structure standard to establish the directory structure and corresponding branching strategy in Helix Core. This is true whether you are migrating from another SCM system or starting new projects in Helix Core.

A template for developing your own directory structure standard is available for download here. You can participate in discussions about the PDS on the Perforce Forums here. Contact Perforce Consulting (consulting@peforce.com) for further information.

### 2.3 Release Processes and the PDS

The directory structure in Perforce Helix Core can be thought of as “low” and “high” levels. Low levels represent your software products and can vary for each software product. High levels of the 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 automation of various kinds.

Migration to Helix Core typically involves defining a directory structure standard for each product imported, and in some cases for the entire organization. A directory structure standard 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 associated with various software products in Helix Core.

For example, one software product might be a licensed software product, the release process for which would define how to maintain old releases and deliver patches. A web-based software product operated in your own data center would follow a different release process, in which there is little need to maintain old releases, but which must support rapid updates. Still another product might be a set of generic components that are delivered to customers and then heavily customized, perhaps by your own professional services organization.

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

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

### 2.4 Addressing Intellectual Property Concerns

Maintaining IP provenance (i.e., knowing where your source code came from, knowing what legal rights you have to it) can be a priority in SCM migration

Copyright © 2008-2019, Perforce Software. All rights reserved.

scenarios. From the perspective of a migration process, your goal should be to ensure that IP provenance is not negatively affected by the migration. Your migration processes should provide a clear audit trail so that all imported files can be traced back to the original legacy repository.

SCM 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 your legacy SCM system.

Migrations provide an opportunity review access control policy. In some cases, ensuring strong IP protections requires extra effort, causing people to wonder if strong access controls really benefit the organization. In other cases, migrations expose particularly weak access controls, and Helix Core's powerful and flexible access control capabilities can be taken advantage of to provide a straightforward means of guarding IP with relative ease.

### *2.5 Training*

Training for Helix Core users and administrators is essential to help a migration go smoothly, and to help get the most from Perforce after the migration. With respect to scheduling, we find it most effective if training for the bulk of users occurs ideally a few days prior to the cutover to Helix Core.

For information on training options available from Perforce, see:
[https://www.perforce.com/support/training](https://www.perforce.com/support/training)

### *2.6 Perforce Transition Team*

We recommend establishing a transition team. This core group may include application administrators, system administrators, and other influential users. You might consider engaging Perforce Consulting to participate in your transition team.

The transition team defines how Helix Core will be used in your organization, how it will tie into your various business processes and workflows, and how it will be integrated with other systems such as Perforce Hansoft.

For larger migration 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 proliferated during the training for the larger user community, which occurs later in the migration process.

## 3 Import Strategies

There are three approaches for importing files into:

- Starting over (tips),
- Detailed History Import (DHI), which can be exhaustive or selective, and the
- Baseline & Branch Import (BBI) strategy.

Copyright © 2008-2019, Perforce Software. All rights reserved. 

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

## *3.1 Tips – Starting Over*

This isn't really a conversion strategy. This approach is to get the "tips" – the latest file versions from your legacy version control tool, and simply add them to Helix Core, without any history at all.

Based on the organization's directory structure standard, a high level directory is identified in which the files will be stored, perhaps something like:

```
//Eng/Gizmo/MAIN/src
```

Here, "Eng" is for Engineering, Gizmo is a product name, MAIN indicates files in the main stream of development, and 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 things like documentation VOBs, or VOBs for shelved (but not terminated) projects. It 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 mapped correctly, and file type modifiers are applied, such as '+x' for executables, '+F' for compressed binary formats like *.gz or *.mpg, etc.

### 3.1.1 Starting Over - Pros:

- It is easy. You need only define target directories in Helix Core, and then add the files.
- It is fast.

### 3.1.2 Starting Over - Cons:

- No historical information is available in Helix Core.
- If multiple branches are imported, Helix Core won't be able to simplify first-time merges between the branches, as that requires knowing the historical relationship among the branches.

## *3.2 Detailed History Import (DHI)*

This is the logical extreme case of conversion. The goal with this approach is to capture as much legacy SCM information as practical, so that comprehensive historical research can be done in the new system, with the old system being taken offline entirely.

Published tools exist to import to Perforce Helix Core from Git, Subversion, CVS, PVCS, RCS, Visual Source Safe. Unpublished DHI tools for migrating from some other legacy systems exist, including IBM Rational ClearCase™ -- contact Perforce

Copyright © 2008-2019, Perforce Software. All rights reserved. 

Consulting for details. Consulting has been involved in migrations from many other tools.

In other cases, conversion tools can be developed to extract legacy SCM data and create roughly equivalent Helix Core data. Because SCM systems vary greatly in architecture, functionality, and what data they store, there are limits on exactly what "detailed history" can be imported. At the very least, contents of file versions at each file revision, and associated metadata are preserved including userid of the submitter, checkin comments, and timestamp of checkin. More sophisticated approaches may also capture branching and integration history, track renames, etc.

### 3.2.1 Pros:

- Comprehensive historical and "code forensics" research can be done without the benefit of the old system, which can be taken offline if necessary.

- Once in Helix Core, you can use powerful Helix file and directory diff in the P4V (visual client), the P4V Revision Graph, and P4V Time Lapse view to see your old files in a new light.

- Helix Swarm can be used to initiate code reviews on older code changes, even those made years ago in the old system.

- There is increased benefit for systems integrated with version control. For example, the meaning of the linkage between a set of files originally modified in your legacy SCM and an issue from your issue tracking system can be maintained.

- Once historical data is in Helix Core, it will gain the benefit of checksum verification of contents of all revisions, improving IP provenance. Unlike Helix Core, most legacy SCM systems do not have a way to validate the integrity of versioned file contents or metadata using checksums. Corruption of file contents, e.g. due to disk failures, can go undetected[1].

### 3.2.2 Cons:

- Complexity of migrations translates into potential schedule and budget risks if snags are encountered.

- Hardware capacity planning is impacted. Any SCM system with say 7 years of history could be expected to require more hardware (more disk space, more RAM, faster CPUs and I/O subsystems, etc.) than one with no history. If you do a detailed import of 7 years of history, your fresh new system will still

[1] Often corruption of older, infrequently used file versions is first detected when migrating from legacy SCM systems. Such scenarios are occasionally encountered when migrating from particularly old CVS, Subversion, and ClearCase repositories.

Copyright © 2008-2019, Perforce Software. All rights reserved.

have 7 years of history, and will initially require as much hardware as if it had it been in operation for 7 years.

- Detailed history imports may require temporary allocation of powerful hardware to support the migration effort. Import tools are not always efficient, and have resource needs typically much greater than a nominal operating Helix Core server would require, sometimes requiring massive amounts of temporary disk space (e.g. 20x greater than the original data).
- Even proven detailed history import tools might choke on your data set. This might be true if the data set is unusually large or contains unusual data patterns or even data corruption.
- Detailed history imports generate significant Helix Core metadata, potentially excessive amounts.

### 3.3 Baseline & Branch Import (BBI)

The baselines & branch import (BBI) strategy provides a lightweight migration alternative that is far more sophisticated than the simple Start Over approach, yet without the technical complexity, schedule and budget risks involved in detailed history imports.

The baseline & branch import process is a generic from-anything-to-Helix Core process, and has been used to migrate to Perforce Helix Core from a variety of SCM systems, including IBM Rational ClearCase®, Borland StarTeam®, Merant PVCS®, Subversion, CVS, Microsoft Visual Source Safe, Microsoft TFS, AccuRev, Perforce Surround SCM, and even unsophisticated SCM "systems" like a set of network drives with directories named to indicate releases.

With the BBI approach, the "interesting history" to be imported is described in the form of a branch diagram that shows the *baselines* (snapshots of a directory structure at a point in time) and major branching operations. For example, a diagram like the following might represent a software product:

Copyright © 2008-2019, Perforce Software. All rights reserved.

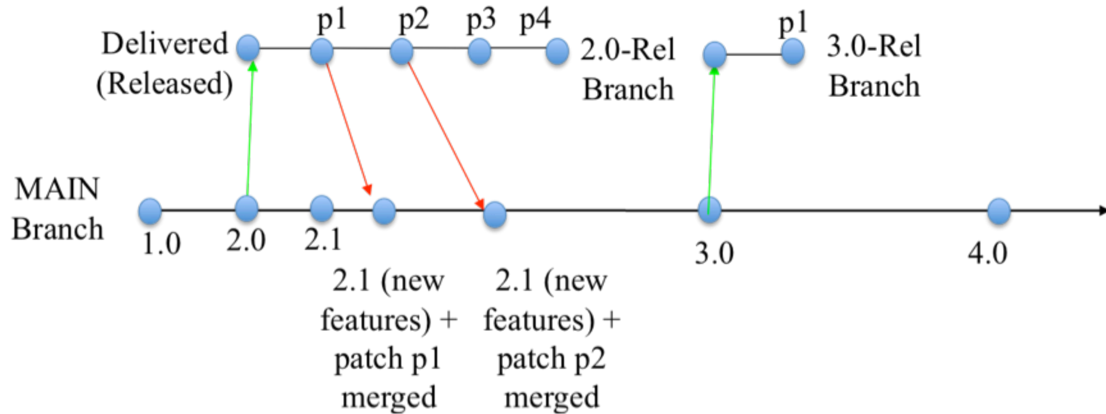


**Figure 1: Sample Baseline & Branch Diagram**

The baselines (blue dots) indicate what "interesting versions" are to be imported. The arrows indicate major branching operations – that is, branching operations that affect an entire branch. In this scenario, a 2.0-Rel branch has been created, and four patches were created on that branch. As of the time of cutover to Helix Core, only two of those four patches have been merged back to MAIN. The BBI process imports all the baselines, records the fact that the merges of two patches were completed with resulting updates to MAIN, and tracks the two unmerged patches remaining on the release branch. Once in Helix Core, its powerful integration engine can be used to complete those merges.

Importing the branching operations allows Helix Core to select common ancestors when doing merge work, thus allowing you to pick up where you left off with branching activities, after the cutover to Helix Core. The BBI process imports branching operations at a high level, capturing the sum of merge operations. For example, in the diagram above, the arrow representing the merge of p2 back to MAIN would likely have occurred as a series 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) is tracked. The imported baseline represents a point in time when the merge of p2 is considered complete.

The intent is to bring over just enough branching history to answer key questions, like "What did Release 2.0 look like?", "Where was this file branched from?" and "What files do I need in my workspace to start maintenance work on Release 2.3?" The BBI approach preserves file contents at key points, and preserves enough branching history so that cutover to Helix Core can happen at any point in the release cycle, rather than just at "convenient" points in the schedule (which tend to be hard to find).

Copyright © 2008-2019, Perforce Software. All rights reserved.

Thus, after conversion, Helix Core would show history of your software product in its Revision Graph tool that would essentially be similar to what would be shown had development occurred in Helix Core to begin with. Detailed data is lost – you will know what the state of your product looked like at Release 1.0 and Release 2.0, for example, but the details of the many hundreds or thousands of checkins between those baselines are discarded, such as the userid, date, time, checkin comments, and contents of each incremental change.

Baseline history diagrams are essential for planning a BBI migration. Ideally, release engineers can quickly draw a branch history picture that captures the desired set of baselines to import for each software product to be imported. These diagrams capture the intended representation of history in Helix Core. In some cases liberties may be taken with actual history to simplify the representation of files as imported into Helix Core.

If it is not the case that diagrams can be extracted from the memory of human admins, such information can be extracted by exploring the legacy SCM system. Once the baseline history diagram is drawn and vetted by key people, it is translated into a set of Helix commands that replay the high-level history in Helix Core. The first baseline will appear as an initial addition of the entire product directory tree. Subsequent baselines result in Helix changelists that show only the changes (files added, deleted, or modified) between baselines. Branching operations are translated into Helix equivalents. Merges done in the legacy SCM system are recorded in such a way that Helix Core honors the results of the merges done in the original system.

If detailed historical research is often needed, the legacy SCM system can be kept online (perhaps with a single license). It is a good idea to keep the legacy SCM system around for a year or two after a BBI migration.

### 3.3.1 Pros:

- A migration from multiple legacy SCM systems used by different teams is straightforward, with each team going to Helix Core on their own schedule and without unduly impacting others. Because the BBI approach works against a live, running Helix Core server (rather than generating separate Helix Core server instances like some detailed history import tools), the project planning for the various teams does not require coordination. Each team can migrate to Helix Core without impacting those already using Helix Core.

- "Interesting" history is available in Helix Core. Once imported, you can use powerful file and directory diff tools in P4V (the visual client), the P4V Revision Graph, and P4V Time Lapse view to see 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 what how the software product evolved from baseline to baseline.

Copyright © 2008-2019, Perforce Software. All rights reserved.

- The BBI process is fairly straightforward with little risk of technical snags.
- Compared to detailed options, this approach makes the data migration aspect particularly easy, because all the historical information can be loaded into Helix at any point in time prior to cutover. Then, on the day of cutover, only the baselines representing latest state of development on active branches need to be brought into Helix, as all historical information would already be imported.
- BBI runs very quickly, so throw-away dry runs can be done in order to develop and test any source code changes that may need to be done as part of the migration, such as updates to build scripts or makefiles.
- The amount of metadata in Helix resulting from a BBI is negligible; it does not unduly impact performance or initial capacity planning.
- You have the opportunity to normalize past history into a new PDS, which indicates activities such as creation of branches in a consistent manner. In cases where branching strategies evolved over time with the legacy SCM system (or even over a series of legacy SCM systems), this provides a chance to simplify historical research of the imported baselines. With the BBI approach, it can be made so that 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 Cons:

- In cases where files were renamed or directory structures reorganized between releases, the historical connection between the files in the old name and the new name are difficult (but possible) to capture. For example, if a 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. That historical linkage of the renaming is often forgone in BBI migrations. With additional effort to identify refactoring events and layout baselines in such a way that from/to information is identified, renaming can be captured properly, as opposed to simply showing a delete of one file and an add of another without the connection that the two are related. However, detecting and handling renames with a BBI import requires substantial effort, and is often impractical.
- If your software product is built with complex component-based development methodologies, or otherwise has complex dependencies, getting all the required baselines imported to be referenced can be a complex challenge. In extreme cases where arbitrary versions of other imported software must be referenced, a BBI approach may approach or surpass the complexity of a DHI approach.

Copyright © 2008-2019, Perforce Software. All rights reserved.

- If detailed history research must be done, the old SCM system must be kept online (perhaps in read-only mode, and perhaps with minimal “life support” licensing from the vendor). Over the long term, the ability to support or use old systems may diminish with staff turnover.

Copyright © 2008-2019, Perforce Software. All rights reserved.