

Perforce Software, Inc.

510.864.7400

[consulting@perforce.com](mailto:consulting@perforce.com)

[www.perforce.com](http://www.perforce.com)

Migrating to Perforce Streams

Table of Contents

[1 Executive Summary 1](#_Toc420938659)

[2 Prerequisites 1](#_Toc420938660)

[2.1 Altered Directory Structure 1](#_Toc420938661)

[2.2 Branch Stability and Use 2](#_Toc420938662)

[2.3 Stream Composition and Inheritance 2](#_Toc420938663)

[3 Seeding the Streams: How to Move 3](#_Toc420938664)

[4 Impact on Users 5](#_Toc420938665)

[5 Impact on Release and Project Management: When and Who to Move 6](#_Toc420938666)

[6 Impact on Other Tools 6](#_Toc420938667)

[7 Impact on Administration 7](#_Toc420938668)

[8 Conclusion 7](#_Toc420938669)

[9 Resources 7](#_Toc420938670)

# Executive Summary

Perforce streams are an innovative way to manage concurrent development, dependencies, and other common branching and release activities. For new projects, using streams from day one will provide a workflow framework based on best practices observed over many years. Streams are flexible enough to accommodate many branching and development models.

Moving an existing project in Classic Perforce into streams requires some planning. Although the mechanical aspects of moving the data into a stream depot are straightforward, other areas must be addressed, including the impact on users and other tools.

This document highlights several preparatory steps that should be considered before moving an existing project to streams. Although some planning is required, moving to streams does present the chance to take advantage of a productive workflow and address legacy bottlenecks.

# Prerequisites

This document assumes a basic understanding of Perforce Streams: what they are, how they work, and the associated commands and tools. Several references are provided later.

You should also have a firm grasp of the mainline branching model that underpins Perforce streams. Check the Perforce video library [Streams tutorials](http://www.perforce.com/resources/tutorials). See the section “Using Streams to Simplify Codeline Management”. Also useful is the [Perforce Directory Standard](http://www.perforce.com/blog/100607/perforce-directory-standard-pds), which has now been updated to cover streams. Some key streams concepts are covered in the following sections, as they directly impact migration planning.

## Altered Directory Structure

Prior to streams, directory structure was used to convey important contextual information about the intended use and relative stability of branches. For instance, consider the *Jam* project, which has a main branch and two release branches. A common (if not ideal) directory structure is shown in Figure 1, where a *REL* container directory indicates that the sub-directories are release maintenance branches.

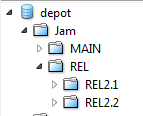


Figure 1: Jam branches

After moving the *Jam* project to streams (and adding a couple of new branches), the directory structure is flat:

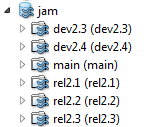


Figure 2: The Jam branches as streams

## Branch Stability and Use

Referring back to Figure 1, the *REL* container level conveyed important information about the branches in that container. These branches were release maintenance branches, implying a higher level of stability than development branches or the main branch. According to the mainline model, a release stream followed a particular flow of change pattern, governing when changes were merged back to the main branch. With streams, this information is captured in the stream metadata and presented visually in the Stream Graph.

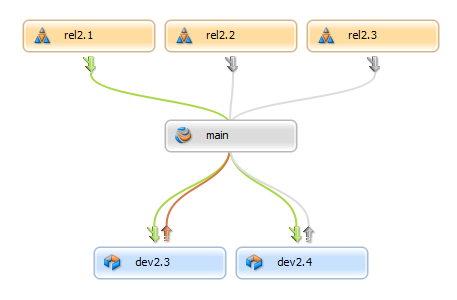


Figure 3: Stream Graph conveys branch stability and intended flow of change

## Stream Composition and Inheritance

Streams, like workspaces and branches, have views. A stream view defines which files are actually branched for work, which are imported or excluded from the parent, and which are imported from other parts of the repository. In other words, a product architect can use the stream view to define the set of modules or components in a stream.

The stream view is inherited by all child streams and workspaces, which simplifies the start-work process for new project users. A new user would create a workspace from a stream, and the workspace view is generated automatically. Workspace views are also updated automatically when moving from stream to stream.

# Seeding the Streams: How to Move

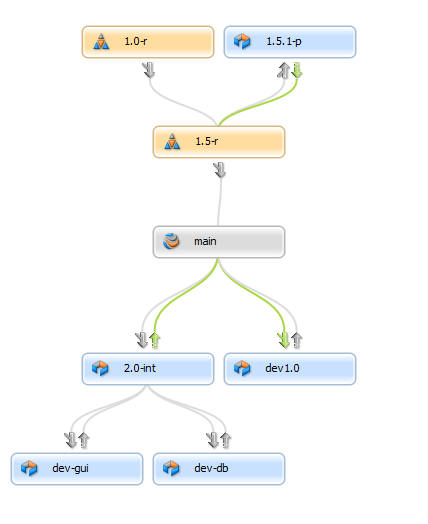
Mechanically moving data into a stream depot for an existing project is straightforward. Although a BBI[[1]](#footnote-1)-style approach could be used, in most cases it will be sufficient to simply integrate (copy) the tips of the relevant branches into new streams. Perforce will, of course, track and respect the indirect branching history between the new streams.

At a high level, the steps involved would be:

* Define a new stream depot for the project (administrative access required).
* Choose the relevant branches to copy to the stream depot. All branches could be included, or only those branches that are still actively used.
* For each branch, define an equivalent stream.
  + Start with the *main* stream.
  + Choose stream names carefully. Remember that the stream depot directory structure is flat. The stream metadata captures a lot of the important information about streams, but a good naming convention will help your users.
  + Consider the type of stream to use. *Release* streams are assumed to be more stable than the parent; *development* streams are less stable than the parent. A *mainline* stream is, of course, the main branch; there is usually one mainline per stream depot.
  + Consider the allowed flow of change (set in the stream definition). Most development branches allow a bidirectional flow of change; release maintenance branches usually do not accept changes from the parent.
  + Carefully determine the parent-child relationship between the streams. At first glance this appears an easy task, but there are practical implications to consider. For instance, if you normally merge bug fixes from oldest release to newest release and on to the mainline, you may want to have the oldest release use the next oldest release as its parent, and so on. In other words, take into account the way that you normally perform merges between branches. Compared to the very simple structure in Figure 3, Figure 4 shows a more advanced branch model, with an older 1.0 release receiving bug fixes first, followed by a 1.5 release and then the main branch.
  + Capture any relevant information about stream composition in the stream views. This information may be currently stored in a branch spec, or may be implied. For instance, in Figure 4 the *dev-db* stream imports two modules from the integration stream, as shown in the stream paths.
* Copy each branch from its original location to the equivalent stream using the *p4 copy* command. For example:

p4 copy -v //depot/Jam/MAIN/... //jam/main/...

p4 submit –d “seeding Jam stream mainline”



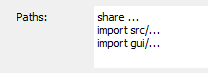


Figure 4: A more advanced branch model in the Stream Graph

The Revision Graph for one file shows that the pre-streams history of the file is readily accessible:

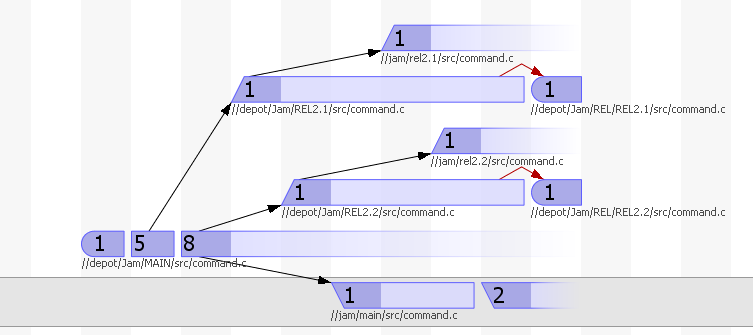


Figure 5: Stream Graph showing streams and pre-streams history

In most cases, due to the indirect history between the newly copied streams, merging between two streams will give similar results as merging between two of the original branches. Going forward the streams may diverge, of course, but after copying them from the legacy location the merge history will not change.

As an example, perhaps there is currently one bug fix waiting to be merged from the Jam *REL2.1* branch back to *MAIN*. After integrating the current state of *MAIN* and *REL2.1* to streams, running p4 integ -S //jam/rel2.1 will also show that pending bug fix.

In cases with more complex merge history, it would be wise to preview any merge operations immediately after moving to streams, to ensure that the results are as expected.

# Impact on Users

Using streams will be easy and straightforward for most users. However, using streams is a change for those used to classic Perforce branching. Training and documentation should be provided in advance.

Of particular interest will be the new, simplified workflow that streams supports, and the related commands or GUI functions. For example:

* Simplified workspace creation and management
* Stream (branch) creation
* The merge down/copy up paradigm (supported by the *merge* and *copy* commands)
* Stream view management
* In-place branching (fast workspace switching)

Aside from the new workflow and commands, the decision to start working in streams will have other direct impacts on the users. They will need new workspaces, and other tools and scripts may need to be updated accordingly. They will also need to understand the new directory location and stream structure.

# Impact on Release and Project Management: When and Who to Move

Streams will simplify many release management activities. The flow of change is guided by the streams framework, and the stream view allows for dependency management and code re-use. Build and release scripts and tools will need to be updated to take advantage of these new features.

If the previous branching model was well laid out, with good supporting scripts and tools, then the overall workflow will feel familiar. An inefficient branch and release model will become more visible in the streams framework; users will find themselves working outside of the expected guidelines frequently. Moving to streams is thus a good opportunity to identify and fix any problems in the legacy model.

Another important choice is how many teams to move over, and when they move. Moving over the entire organization to streams at once simplifies some of the planning, since legacy tools and processes could be retired. However, that would require very careful planning and management.

Moving over a pilot team, and then moving over additional teams incrementally, allows more time for new processes and tools to be fleshed out. If there are strong dependencies between the work done by different teams, additional work will be required so that the team using classic Perforce can still collaborate with the team using streams, and vice versa. The migration schedule for various teams will thus depend greatly on the level of collaboration, schedule of releases and other important milestones, and the ability of each team to adapt to new processes.

In the course of planning, you may find that some teams should not move to streams. For example, teams that simply use Perforce as a document repository, with little or no branching and parallel work, would not benefit much from the streams workflow. Other teams may already have a comprehensive set of scripts and tools to support their unique development process, in which case moving to streams may prove disruptive. Streams provides a built-in workflow based on observed best practices; if your current needs are well met without streams, moving to streams is not necessary.

# Impact on Other Tools

Other application lifecycle management (ALM) tools interact heavily with Perforce. Code review tools, defect trackers, and continuous integration (CI) engines all interface with Perforce to some degree. Using streams will impact these tools, and that impact must be analyzed and understood in advance.

Every tool that relies on knowing and understanding the product directory layout needs to change. Of course streams are in a different location, but the directory structure is now flat; tools and scripts cannot rely on the directory conventions that used to convey structural context. Streams capture the same information in other ways, so the tools that rely on this information must change.

# Impact on Administration

Using streams will most likely require the creation and use of more depots (one per product) than before. Creating these depots and setting the appropriate permissions will be an administrative task.

Using streams requires a 2011.1 or later Perforce server and streams-enabled clients. Administrators will need to manage a server upgrade and make sure that all affected users have appropriate client software.

# Conclusion

This document details, at a very high level, some of the considerations involved when moving existing users and projects to Perforce streams. The key to any process transition is planning and preparation. Moving to streams is no exception, particularly if you want to realize the potential productivity gains.

For further information or assistance, please contact Perforce Technical Support or Perforce Consulting.

# Resources

|  |  |
| --- | --- |
| Perforce Technical Support | [support@perforce.com](mailto:support@perforce.com) |
| Perforce Consulting | [consulting@perforce.com](mailto:consulting@perforce.com) |
| Perforce Directory Standard | <http://www.perforce.com/blog/100607/perforce-directory-standard-pds> |
| Introducing to Streams | <http://www.perforce.com/blog/110308/introducing-perforce-streams> |
| High level stream overview | <http://www.perforce.com/blog/110317/streams-big-picture> |
| Stream tutorial | <http://www.perforce.com/blog/110324/streams-tiny-tutorial> |
| Streams and the flow of change | <http://www.perforce.com/blog/110419/streams-flow-change> |
| Stream views explained | <http://www.perforce.com/blog/110517/streams-taking-you-out-views> |
| Streams applied | <http://www.perforce.com/blog/110526/streams-applied-mainline-development> |

1. Baseline and Branch Import, a data migration technique that captures important milestones and the branching relationship between them. [↑](#footnote-ref-1)