

Mitigating the Risks of “Deep” Renames in the Depot

Introduction

At Docent, we follow a common development practice of maintaining a *mainline* of development in one source tree in our Perforce depot. Releases, patches, and special projects are handled in branched trees, mapped either to the *mainline* or to one another through Perforce branch views. This practice has been flexible enough to handle most of our software development needs.

However, an exceptional situation developed. It comprised a completely new generation of software product, written mostly from scratch. The architecture was new; the source tree layout was new; the use of third-party tools differed. Overall, the differences were so great that it was unnatural to think of this new project as anything that had branched from or would integrate back into our current *mainline* of development. So when this project matured and gained the confidence of our organization, everyone expected that it would somehow become the new *mainline* of development for our future products.

As a rule, whenever the *mainline* of development gets supplanted by an unrelated source tree, it may be appropriate to *promote* the new source tree to become the *mainline* after *demoting* the former *mainline* source tree to some historical name.

This paper describes the objectives, risks, preparation, and mechanics of doing a “deep” rename of source trees in a Perforce depot. The rename is “deep” in the sense that it runs across all file versions through time and makes the renamed source tree appear to have been developed all-along bearing its new name. In contrast, a “shallow” rename would manipulate only the head revisions of files using paired integrate and delete commands in Perforce. Since a “deep” rename entails editing metadata in the Perforce depot, it is generally considered risky.

This paper also describes an associated organizational problem and its solution. The rename task had to be accepted by the community of Perforce users who had concerns about impacts, risks, and timing.

Intended Audience

If you think people should work around the limitations of tools and you think the trouble and risk entailed in changing some silly names in Perforce are a waste of time, then this paper is probably not for you. If however you think tools should serve organizational policy and you think that good names and notation are critical to clear thinking about engineering problems, then this paper might be useful to you.

Objectives and Reactions

When applied to engineering projects, the emotive connotations of the words *promotion* and *demotion* were appropriate in our situation at Docent. While the new project was maturing, our current *mainline* source was quite actively spawning minor releases. There were two corresponding engineering teams, and both were busy. At the time of the promotion of the new project, we were not at all intending to kill or delete the current *mainline* source. My proposal to rename these source trees elicited contrasting emotional reactions from members of the respective team. Both teams wanted to think of themselves as working on *mainline* source, at the same time. Aside from the mechanics of any Perforce rename, this situation posed the associated organizational problem to be solved.

The *promotion* was popular with the new project team. It served their purposes. They wanted:

1. organizational focus on the project,
2. their own name space for third-party tools,
3. formal branch views to designate cross-generation relationships between the current release source and their new project source,
4. segregation of unrelated source code well away from current source with which it might be confused,
5. early stability in the project’s name space, especially for new engineers creating new client views of the depot.

They argued for the renames to be done *sooner* rather than later.

The *demotion* was objectionable to the current release team. That team wanted:

1. continued organizational focus on current releases,
2. no disruption or risk to current schedules,
3. no disruption of current habits,
4. no disruption of current name spaces.

They tacitly indicated morale would suffer on current engineering projects if all their source were *demoted* to make room for an incomplete, relatively untested, new project. They argued for the renames to be done *later* rather than sooner.

I added an objective of tolerating “open edits” in client workspaces during the rename work in the depot, such that Perforce users would not be required to submit all changes to the depot prior to the shutdown. I did not want to create a situation that would either cause premature submissions of untested changes to go into the depot or cause developers to discard partly-implemented but worthy engineering efforts.

Resolution

Organizational Solution

I accommodated the conflicting demands of the two engineering teams by inventing an ambiguous name containing the word “main” and then restructuring my rename task in the depot. In place of an instantaneous rename, which would impact both teams, I scheduled two renames with a long delay between, which would impact one team first and the other much later. Although this forced two scheduled Perforce shutdowns in place of one, the delay allowed development to go forward for several months on two source trees with the word “main” in their names. Ambiguity enhanced morale. In practice, the *promotion* occurred early and the *demotion* much later. Only when activity had nearly ceased in the older source tree, was it renamed from “main” to its *demoted* historical name.

Mechanical Solution

The decision to do “deep” renames rather than “shallow” was made because of experimental results. At first there was no choice, since Perforce documentation only describes renames of head revisions, which I call “shallow”. The alternative of “deep” renames started as a *hope* or hunch on my part that there must be a better way to do the renames such that file histories were preserved and open edits were not problematic. Perforce technical support realized that hope and enabled me to construct further experiments yielding better results. I understand that “deep” renames were undocumented because they were judged too risky to recommend generally.

Mechanics

Experimentally Developing a Plan

In bullet format, these were the steps taken:

- Created a copy of our full depot on a separate Solaris server host as a sandbox.
- Created two client workspaces in the sandbox: one representing a cooperative Perforce user, the other representing an uncooperative user who ignores all instructions and warnings.
- Expected to iterate over proposed methods, using Bourne shell scripts that automated the task, documented the task, and made it easy to adapt actions to circumstances, and provided a method of resetting the depot to a prior state to facilitate repeating the rename task.
- Experimented first with “shallow” renames:
 - wrote corresponding scripts,
 - timed their execution,
 - examined consequences and the structure of the sandbox depot,
 - questioned the convenience viewing of file revision histories,
 - encountered problems with “open edits”.
- Asked Perforce Technical Support about this whole problem area and the concerns my experiments had generated.
- Got excellent advice from Perforce, who described in detail the *deep* rename alternative entailing edits of checkpoint files and corresponding moves in the RCS tree.
- Consequently, wrote alternative shell scripts to do the work of a “deep” rename.
- Did test runs in the sandbox and gained experience resolving problems in that depot and in the representative client workspaces.
- Practiced recovery from partly executed scripts, often chopped off the tail portion of the script that had not yet run and fashioned a custom script to complete the task, adapted the scripts to make this sort of recovery easier, also made the script output verbose so it would be easier to infer exactly where each script failed and what precisely remained to be done.
- Used sandbox checkpointing and script timings to estimate down time for the real depot.
- Parameterized the shell scripts so there were minimal differences between running the scripts on the sandbox depot versus running them on the real depot (to reduce risk and make testing most relevant),
- Practiced restoring the sandbox depot for new test runs and for experience in Perforce disaster recovery. Thus gained confidence in modifying checkpoint files and restoring Perforce databases from them,
- Executed typical Perforce client actions on renamed trees to examine functionality in a modified depot:
 - tested labels and branches,
 - examined file revision histories,
 - studied the “open edits” scenario where files are open for edit by an engineer who is ignoring all messages from me,
 - noted the error messages seen in that scenario and determined recovery actions that would preserve the engineer’s work-in-process.
- Stress-tested the sandbox with the `p4 obliterate` command, because this command operates directly on the depot and relies heavily on internal consistency of metadata.
- Updated the sandbox depot with new copies of the real depot periodically, to keep testing and timings relevant.
- Prepared a template for generating the tailored e-mail messages I would send to our Perforce users individually.

- Studied all existing client workspace views in light of the pending renames and considered the patterns of advice I would give.

Executing the Plan

- Tailored the tested scripts to the real depot, by changing simple parameters.
- Announced the scheduled shutdown of Perforce.
- Did the renames in the depot:
 - shut down Perforce,
 - did a checkpoint of the Perforce database and a backup of the RCS files,
 - ran the scripts that renamed trees in the depot,
 - did a new backup of the RCS files,
 - started Perforce.
- Announced the renamed trees in the depot and the availability of Perforce generally.
- Sent tailored e-mail messages to Perforce users advising them on each client workspace specifically, giving instructions about renaming directories on local file systems.
- Monitored Perforce users generally and answered questions as they arose.
- Archived the pre-rename backups longer than usual.

An Outline of the Deep Rename Script

A "deep" rename script was written, tested, timed, and used. It required write access to depot directories and Perforce superuser administrative access in a client workspace that contained all the files and directories to be renamed. Here is pseudocode:

- Store the new name in a simple variable, so the name is easily changed in the implementation and typos are more easily caught
- Shut down Perforce by killing `p4d`
- `cd` to the `P4ROOT` of the depot
- Create a pre-rename checkpoint file of the Perforce database, using `p4d -jc`
- Use the checkpoint file as input for global edits that generate a post-rename file in a checkpoint format. Use the simple variable reference in all patterns employing the new name
- Remove the existing Perforce database: `rm -f db.*`
- Restore the database from the post-rename checkpoint file, using `p4d -jr`
- `cd` to the depot subdirectory containing the RCS tree
- Perform file system renames corresponding to the patterns of global edits done above, using the same simple variable references for new names
- Start Perforce by invoking `p4d`
- `cd` to the client workspace
- Use `p4 integ -v` to copy any parts of the depot that should be duplicated rather than renamed
- Invoke `p4 submit` and supply a comment about the copied resources
- Edit existing branch specifications and derive new branch specifications from existing specifications, using the output of `p4 branch -o` piped through `sed` into `p4 branch -i`
- Delete any replaced or obsolete branch specifications, using `p4 branch -d -f`

An Outline of the Shallow Rename Script

A “shallow” rename script was written, tested, timed, and abandoned. Perforce remains up and running; the work is done in a client workspace with Perforce superuser access. Here is pseudocode:

- ❑ `cd` to the top of the client workspace or a lower spot that spans the name space covering the intended renames
- ❑ Move directories to be renamed, using commands in pairs: `p4 integ -v foo/... bar/...`, `p4 delete foo/...`
- ❑ Integrate directories to be copied, using `p4 integ -v`
- ❑ If any new names overlap with former names, invoke `p4 resolve -at`
- ❑ Invoke `p4 submit` and supply a comment about renames and copies
- ❑ Edit existing branch specifications and derive new branch specifications from existing specifications, using the output of `p4 branch -o` piped through `sed` into `p4 branch -i`
- ❑ Delete any replaced or obsolete branch specifications, using `p4 branch -d -f`
- ❑ Edit *all* user specifications, using `p4 user -o` piped through one or two `sed` edit patterns into `p4 user -i -f`
- ❑ Similarly edit *all* client specifications, using `p4 client -o`, etc.

An Outline of the Depot Reset Script

A script was written to reset a target depot and its clients to a prior state, either for another round of testing or for disaster recovery. It assumes you have first made a zipped tar file of the depot `P4ROOT` and clients workspaces in their relative positions prior to testing, when everything was in a consistent state, ready for the renames. Here is pseudocode:

- ❑ `cd` to the depot `P4ROOT`
- ❑ Shut down Perforce by killing `p4d`
- ❑ Remove the journal file and database files, `db.*`
- ❑ Remove the entire `depot` subdirectory containing all RCS files
- ❑ `cd` to each client workspace and remove all source files and directories
- ❑ `cd` to the spot in the file where you previously made the zipped tar file
- ❑ Unzip and extract that archive
- ❑ Start Perforce by invoking `p4d`