**From:** Stanton Stevens [sstevens@adobe.com]
**Sent:** Thursday, June 24, 2004 2:32 PM
**To:** 'Perforce-admins@adobe.com'
**Subject:** obliterate letter

I just sent the letter below to Perforce, along with the list of issues I sent out to perforce-admins on 6/10. The point of "The Obliterate Letter" is that Perforce doesn't really support the end-cycle of versioned files, either archiving or obliterating. This is why I am reluctant to say "go ahead and check in nightly builds", for example, they can be something we end up backing up, copying, sorting through in the Perforce database, etc. for years. Some teams are currently checking in a lot of binaries, I am planning to work with these teams (if I haven't already) to set up some kind of policy for dealing with these files when they are no longer needed. Nightly and weekly builds should definitely have some sort of expiration policy. The details about why Perforce tends to grow quickly and continually are in the letter.

Stanton

-----------------------------------------------------------------------------------------------------------------------
Title: The Obliterate Letter
Attn: Michael Shields, Chris Siewald
Topic: Perforce for more than source code
From: Stanton Stevens, Adobe Systems, Inc.
Date: 6/23/04

Here's a story for you. I worked for a company in the early 90's that made 3D medical imaging stations. They could make spectacular 3D images and movies of people's insides, using non-invasive CT and MR imaging. In order to sell these workstations, we had to create interfaces to tape readers, medical scanners, X-ray film printers, etc. from a number of manufacturers. After we sold them for a while, we were very surprised to find that many customers were using them often, but not as 3D workstations! They were delighted to find that for the first time, they could take images on scanner A, print them with printer B, and archive to tape reader C, all via our workstation. They could also view X-rays on-screen for which the films had been misplaced. Our workstation had become a hub through which any of these big expensive pieces of equipment could work with any other, and the basis of a medical imaging network.

Our company's leadership was smart, noted that the market for networking images throughout hospitals could be a much bigger market than the 3D workstations, and expanded its focus. This turned out to be the case, and the resulting product saved hospitals millions in film costs, so it was a success.

So, how does this relate to obliterate in Perforce?

Perforce, without intending to be, is a powerful tool for binary file distribution. It can manage an arbitrary group of files far better than any kind of simple file server can, no matter what policy is in place. In spite of deficiencies it is the best tool I've found for build and release distribution. It has multiple ways to uniquely identify a set of files, a well defined interface from multiple platforms, and even handles niceties like line endings in text files. It has a gui for browsing, access controls, and a built in file tree copy mechanism (sync). It solves a problem it didn't intend to solve, and is widely used for that purpose here at Adobe. I can hardly tell people not to use it for that when it works so well that I use it myself to move folders and files around when ftp, telnet, etc. are too big a hassle. For the user, there is no apparent problem with distributing binaries via Perforce, after all "disk space is cheap" is still the mantra. To put the binaries on a fileserver with a completely different set of access controls and platform connection problems is not a good alternative.

But disk space on a SAN that is fully mirrored, backed up to tape every night, and scrupulously maintained isn't exactly free. And as more and more about-to-be-useless stuff such as nightly build results is checked in, there is more of a burden on the database tables to search through the records. Checkpoint times go up, performance goes down, and storage use grows out of control.

It is also important to be able to version binaries for other reasons. In order to properly replicate a build environment, third-party software with all included binaries should be version controlled. This includes compilers, tool libraries, database components, installers, etc. Versioning binary "images" of Windows systems is very useful. Media files such as jpegs, mpegs, graphic design files, etc. also need much better control than just a location on a fileserver.

Once a project is swollen with labeled nightly build binaries, etc., the only cure is to take the project down and do large scale obliterates. "Perforce down" is not something I ever like to say, especially to Acrobat's around the clock, around the world development and QE teams. Using p4 obliterate, for even a single file, has a number of serious problems.

**1. Obliterate Issues**

1.1 Nothing else can be done while the database locking portion of an obliterate is taking place, even a sync. This can take from 5 minutes to well over an hour. I just ran a test on a copy of a large system: obliterating one file took 1 hour 35 minutes on an otherwise completely idle Sun Solaris server with 4 processors and 8 GB RAM. Attempts to sync hung during the entire time except the first 2-3 minutes. There were no lazy copies. For this and other experiments, I used p4d 2002.2.

1.2 An attempt to obliterate a 1 MB file that has 20 lazy copies tied to it results in increasing space usage 19 MB. This is dumb, the lazy copies should be re-directed to an existing version of the file. Until this is fixed, many obliterates are downright dangerous, running out of disk space is the one way I've seen a Perforce database reliably corrupted. And obliterate is pointless for reclaiming space if even one lazy copy exists. There is no easy way to determine whether an obliterate will free space or consume it. Running it without the -y flag gives output that could be parsed and processed to yield this information, but you still have to lock the database just to get this information.

1.3 Obliterates are slow, one gigabyte per hour of binaries is as fast as I have seen it go. I have several times needed to obliterate 20 GB at a time, sometimes 50-100 GB when moving a depot to a new project. Preparing a checkpoint for perfmerge is another common use of obliterate.

1.4 It takes much discussion before a team is willing to have things obliterated. Especially once I explain to them that it is nearly impossible to get things back with their histories. Often, the longer something has been there, the less certain people are that it is safe to remove. See suggestion 3.1 for an approach to this problem.

1.5 The obliterate operation often spends hours undoing lazy copies to areas that are about to be obliterated themselves. This accounts for the bulk of the time in obliterates that can take more than 24 hours. It also causes huge swings in the amount of storage space needed to do the obliterate. This has to be watched carefully to make sure available space is not exceeded (if you have enough space to risk it).

**2. Why offline obliterates are a poor alternative:**

2.1 It takes server level access, super user is not enough. This level of access is tightly controlled for security reasons. At Adobe, with 70 Perforce instances and as many admins, this means only I can do offline obliterates.

2.2 Offline obliterates still require an obliterate -z against the live server to undo lazy copies. This causes database locking long enough to be a big problem. I just tested p4 obliterate -z (no -y) on 630 files against a large database it locked the database for 7 minutes. The time appears to proportionate to the number of files.

2.3 Space for an entire copy of the database (db files) is required. Also the possibility of "lazy copy bloat" (see item 1.5, above) means that you have to watch space carefully during the operation.

2.4 Areas to be obliterated this way require entries in the protect table to block access during the obliterate operation. This is hard to specify if you are obliterating all but the head revision, or many different deleted revisions.

2.5 Restoring the checkpoint, creating a client spec with exactly what must be obliterated, monitoring the many hour long obliterate, and playing back the journal into the live instance (which also blocks for a while) is a hassle and takes a lot of my time. I also have to monitor carefully to make sure that storage bloat from undoing lazy copies doesn't fill the filesystem, as mentioned above.

2.6 Deleting by hand the files that have been obliterated is dangerous, and complicated if you are not deleting an entire directory tree. Writing a script to parse the obliterate results and delete only the correct revisions is not simple work, and has its own risks.

## 3. Suggestions

Here are some suggestions for how to expand Perforce's abilities beyond source code. They are mostly concerned with obliterate and otherwise dealing with files that no longer need to be in the archive. This is the main problem to solve for files that are more ephemeral than source code. But all of these ideas would improve our situation with space consumption by Perforce, and make the lives of Perforce admins like me a lot easier.

3.1. Archive storage capability
It would be really useful to be able to select a depot or folder for removal and write it out as an archive that could be restored. This is less final than obliterate, and much simpler than setting aside an entire copy of everything. An archive of this sort that could be added to incrementally would be very helpful for build results. Every few months it could be marked complete, copied off to tape, and forgotten about until the unlikely event the files were needed. Some intermediate level between "using database and storage resources" and "gone forever" is needed.

3.2 No downtime for anything.
This applies especially to obliterate, but checkpointing is another major source of downtime and kludgy offline hassles. Perfmerge also requires downtime. Changing how obliterate works involves database subtleties I can't address, but here are some ideas:
a) Obliterates could operate as a sort of low priority background task, making progress only when user requests were not competing.
b) An option to obliterate could defer it until it was done as part of the next checkpoint operation. This synchronizes it with checkpoint downtime, but does not work for offline checkpoints.
c) The archive storage in 3.1 could help with this, as long as it did not itself cause downtime. It could simply made the file unavailable until a complete archive operation finished the cleanout from the database and file storage.

3.3 Obliterate frees space reliably.
When files are removed, space should be reclaimed. Simple, but vital. This relates to item 1.2. Even with the current "lazy copy bloat" scenario, undoing all lazy copies would present less danger if it was interleaved with file removal, rather than file removal happening only after all lazy copies are undone.

3.4 Use binary diffs
Many binaries change only slightly on each submission. The current method of storing a new copy each time wastes space. There are several good schemes for saving binary diffs.

3.5 More flexible "tempobj"
Keeping only the head revision is too much like a fileserver limitation. If there were options for keeping only the last X number of versions, or an age cutoff, it would much reduce the need for obliterates. This too could work hand in hand with an archive mechanism, putting the versions into an archived status automatically.

## Conclusion:

An ideal solution to managing files covers their entire life cycle of creation, versioning, and eventual archiving and obliteration. We would really like to see Perforce expand its abilities to handle the end phase of the life cycle.

And the moral of the story at the beginning: the number of frustrated users of file servers represents a huge market for Perforce to tap into, probably bigger than the SCM market. And Perforce could hardly be better positioned. Go for it!

Stanton Stevens
Adobe Systems, Inc.
206-675-7595