Obliterate

or

Let's Pretend I Didn't Do That

Julian Foad

WANdisco

Overview

- Why obliterate?
  - Reasons fall into three categories
- How to obliterate?
  - Delete files or undo changes
- Alternatives
  - How it's done in CVS and P4
  - How it's done in Svn now

Why?

- Requested...
  - since pre-1.0
  - by many on mailing list over the years
  - by big companies & organisations
- Reasons stated...
  - hide secrets
  - undo accidental large additions
  - remove old data to reduce repos size
  - work flow needs only recent version(s)

Why?

- Alternatives...
  - can do in CVS (crudely), p4, CC, ...
  - in Svn, authz can hide files
- Counter-arguments...
  - is not “pure” version control
    - and “store only latest” is not VC at all
  - is “dangerous” if mis-used (audit trail)

Why? Hide a Secret

- What happened?

  committed a customer's private data in a company-wide repository,

  committed copyrighted material to a public repository.

- What's needed?

  Hide that file first,

  (may require **swift** action),

  then plan the permanent fix.

Why? Recover Space

- What happened?

  added large, generated files,

  imported to wrong repository,

  split the repos,

  a large project became redundant.

- What's needed?

  Reclaim server disk **space**.

  Can be a **planned** event.

Why? Latest Version

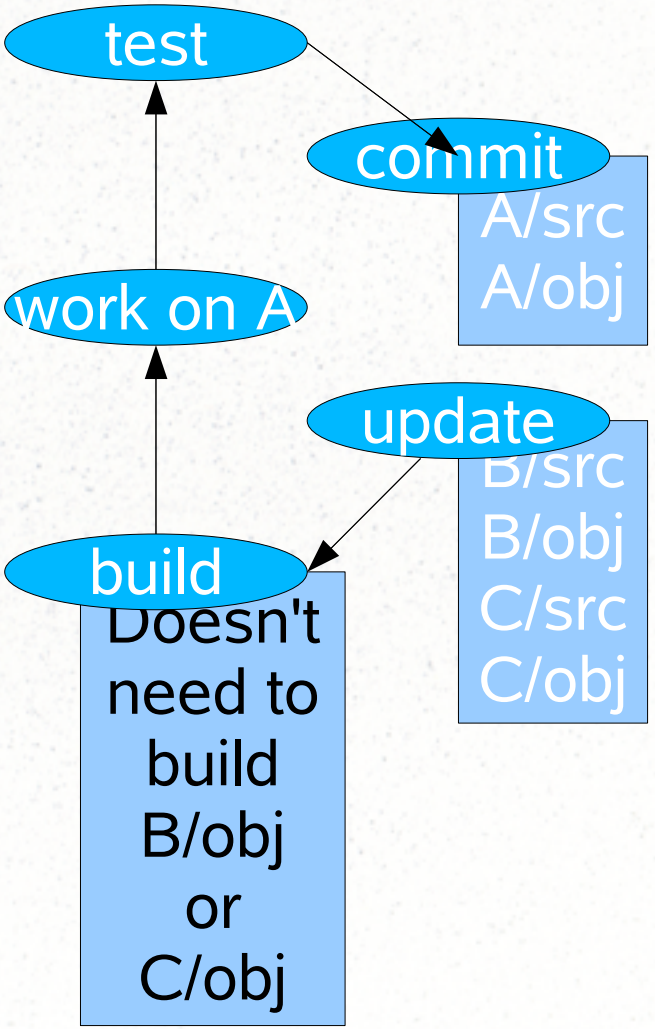
- Project structure:

  moduleA/

  - src/...
  - **obj/...**

  moduleB/...

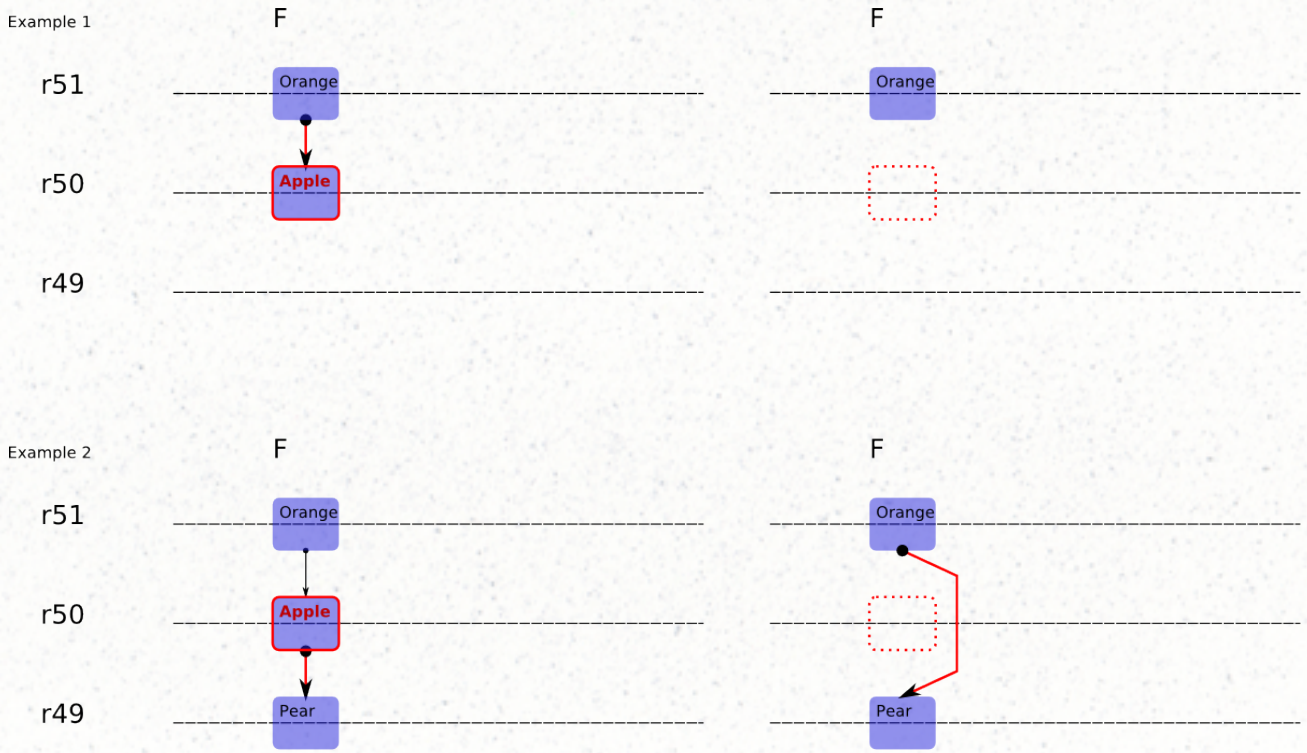
  moduleC/...



Why? Latest Version

- What's happening?
  - frequently revising a large file
  - never need old versions of it
- What's needed?
  - configure to store only latest version
  - make “update” work without deltas
- Why?
  - convenience (could do outside svn)

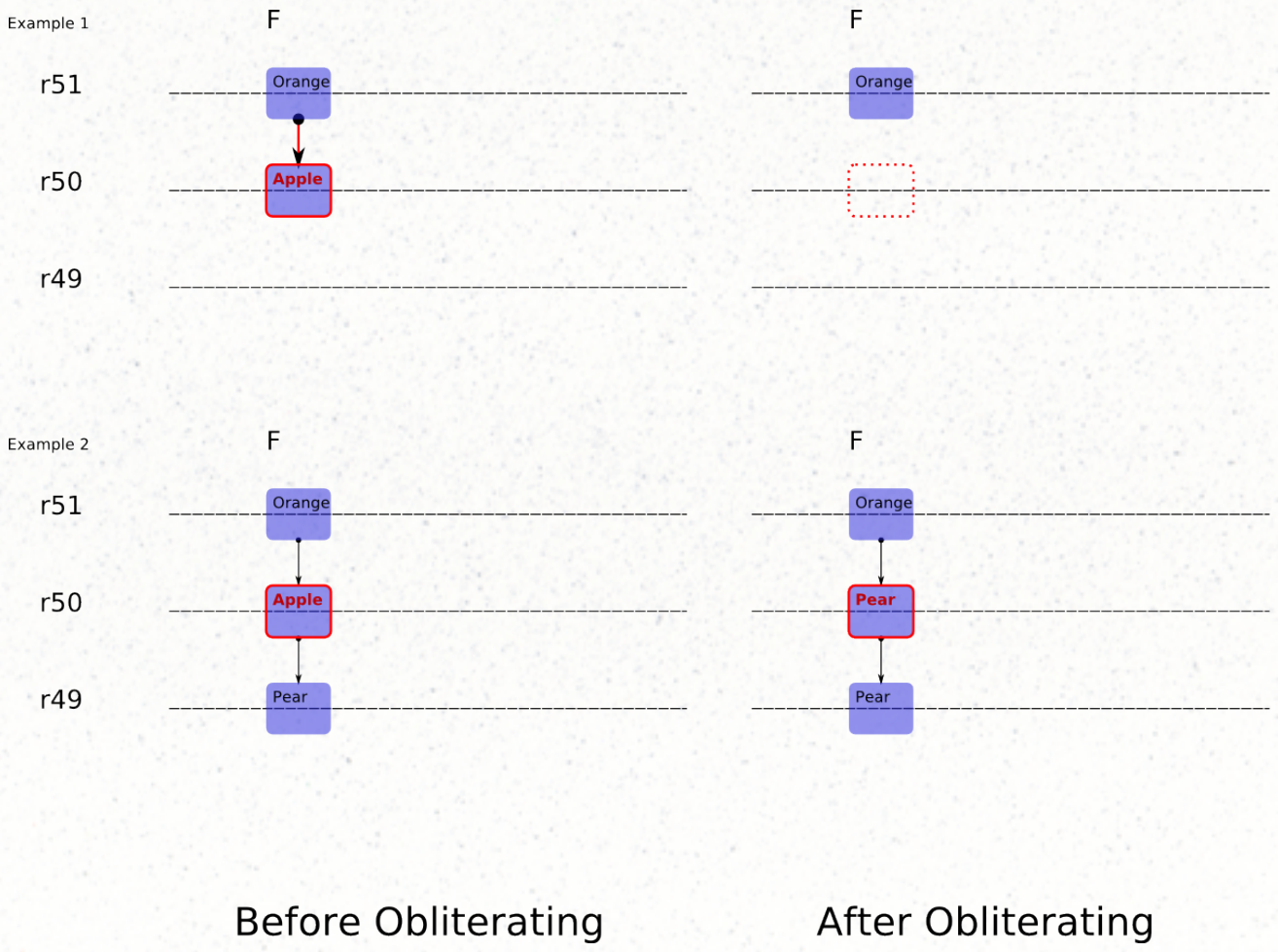
How? Delete Files



Before Obliterating

After Obliterating

How? Undo Changes



What CVS/P4 Users Do

- CVS: Delete the “,v” file (delete a file)
- Perforce: “p4 protect” (can hide file-rev)
- Perforce: “p4 obliterate” (delete file-rev)
- Perforce: “p4 filetype” (latest N revs)
- CC: “rmversion” (delete a node-rev)
- CC: “rmelement” (delete a node)

What Svn Users Do

- dump | svndumpfilter | load
- authz can hide paths

Non-reasons

- These possible reasons to obliterate are not goals
  - to undo (roll back) a recent commit
  - to tidy up history after an unwanted change has been done and undone
- Documentation should steer users to best practices for these tasks

Granularity

- Too big:
  - Remove a whole revision
  - Remove a whole node (or path)
- Seems good:
  - Remove selected node-revs
- Too small:
  - Remove selected text within a file

Obliterate

or

Let's Pretend I Didn't Do That

Julian Foad

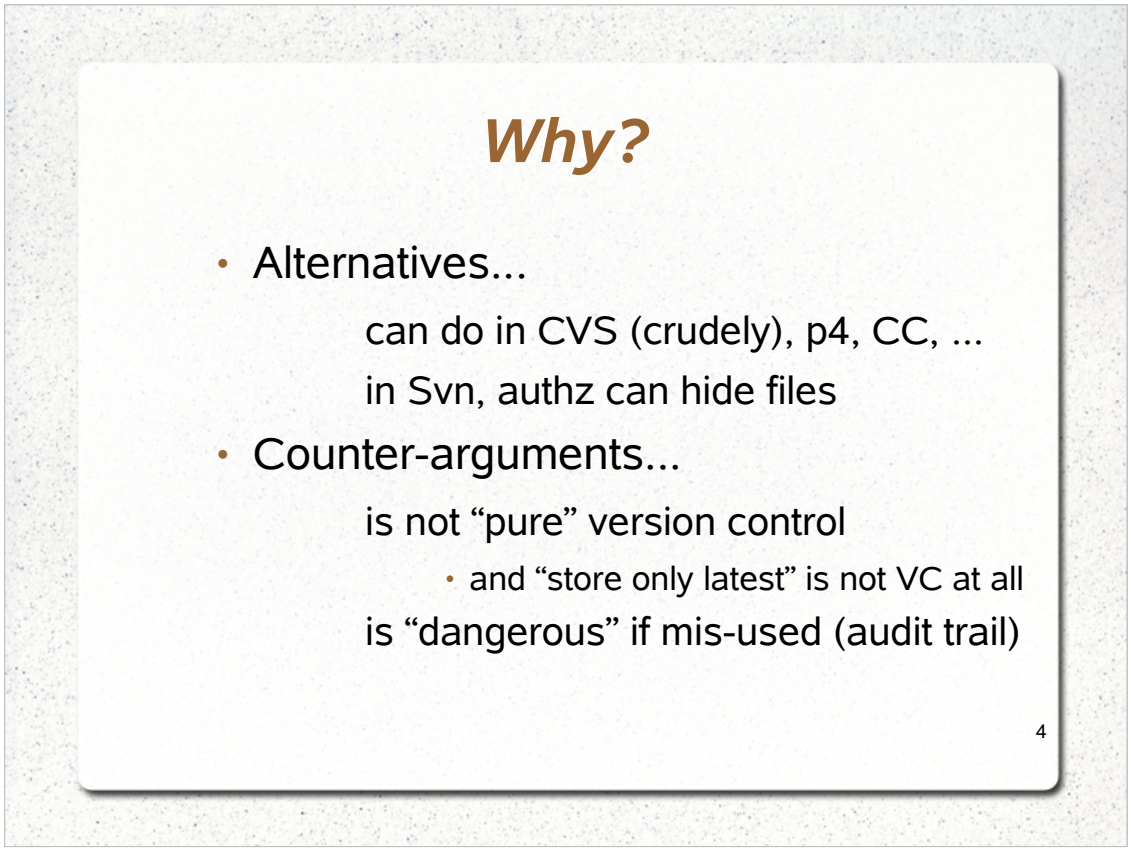
WANdisco

Overview

- Why obliterate?
  - Reasons fall into three categories
- How to obliterate?
  - Delete files or undo changes
- Alternatives
  - How it's done in CVS and P4
  - How it's done in Svn now

Why?

- Requested...
  - since pre-1.0
  - by many on mailing list over the years
  - by big companies & organisations
- Reasons stated...
  - hide secrets
  - undo accidental large additions
  - remove old data to reduce repos size
  - work flow needs only recent version(s)



Counter-argument “not pure VC”:

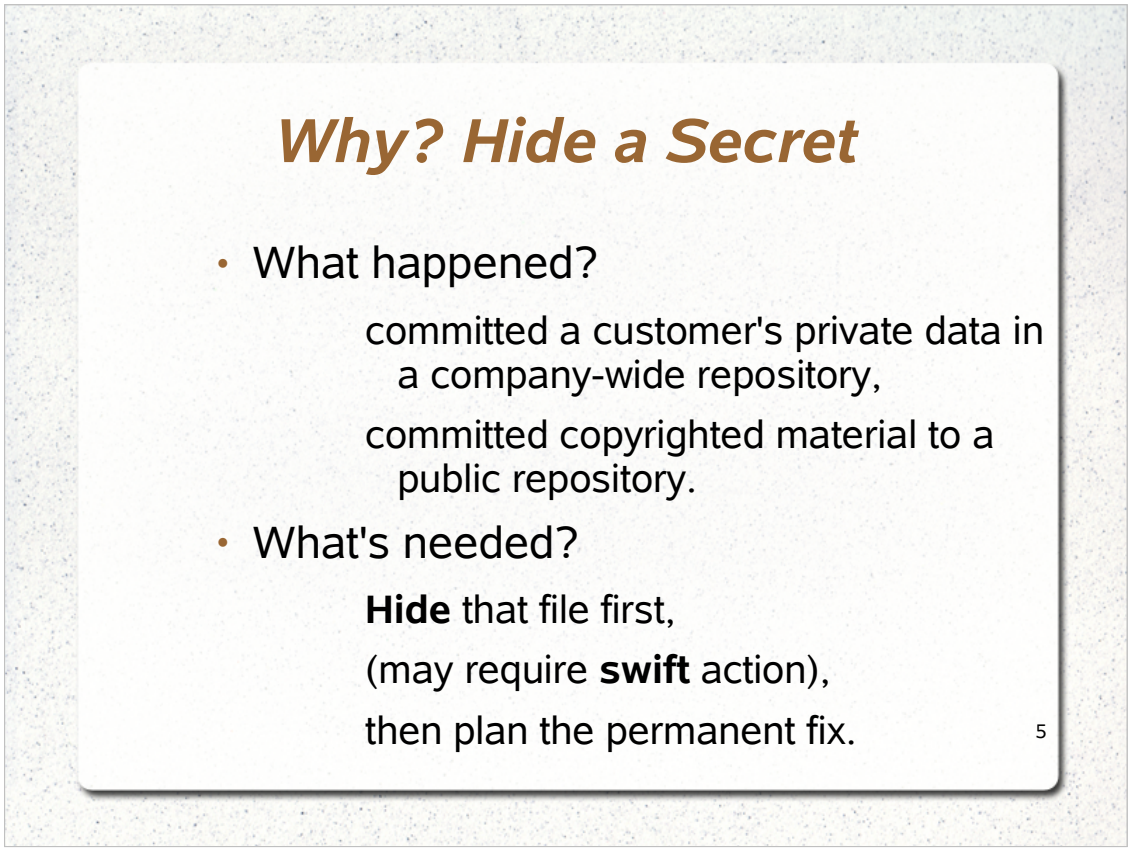
- Any change of history can cause a mismatch between a client's memory and the repository's memory of history, which can lead to client-side difficulties.

- Storing several recent versions can cause the memory-mismatch problems, and storing only the latest version is not VC at all, so do it outside Subversion.

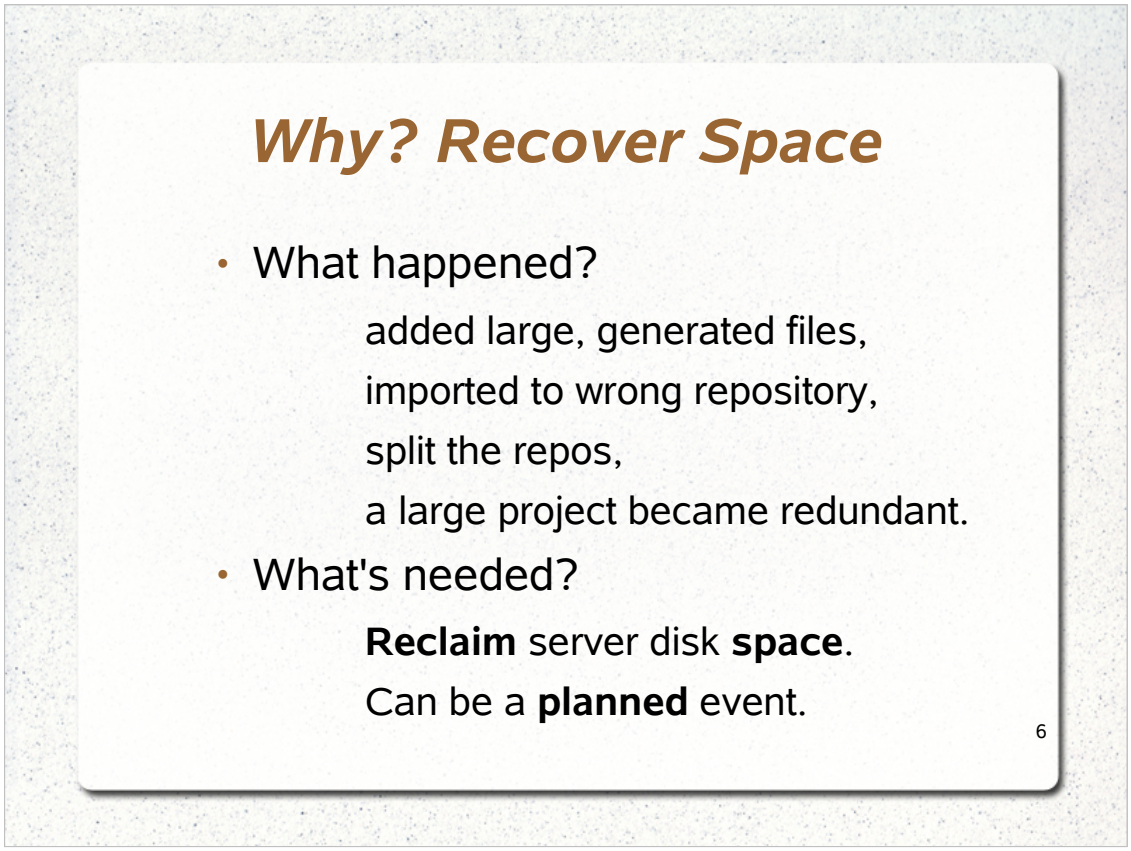
Counter-argument “dangerous”:

- Some people rely on repository history as their audit trail, and the mere possibility of it being changed is a threat.

Both arguments are trumped by practical advantages and solved by pragmatic and responsible usage.

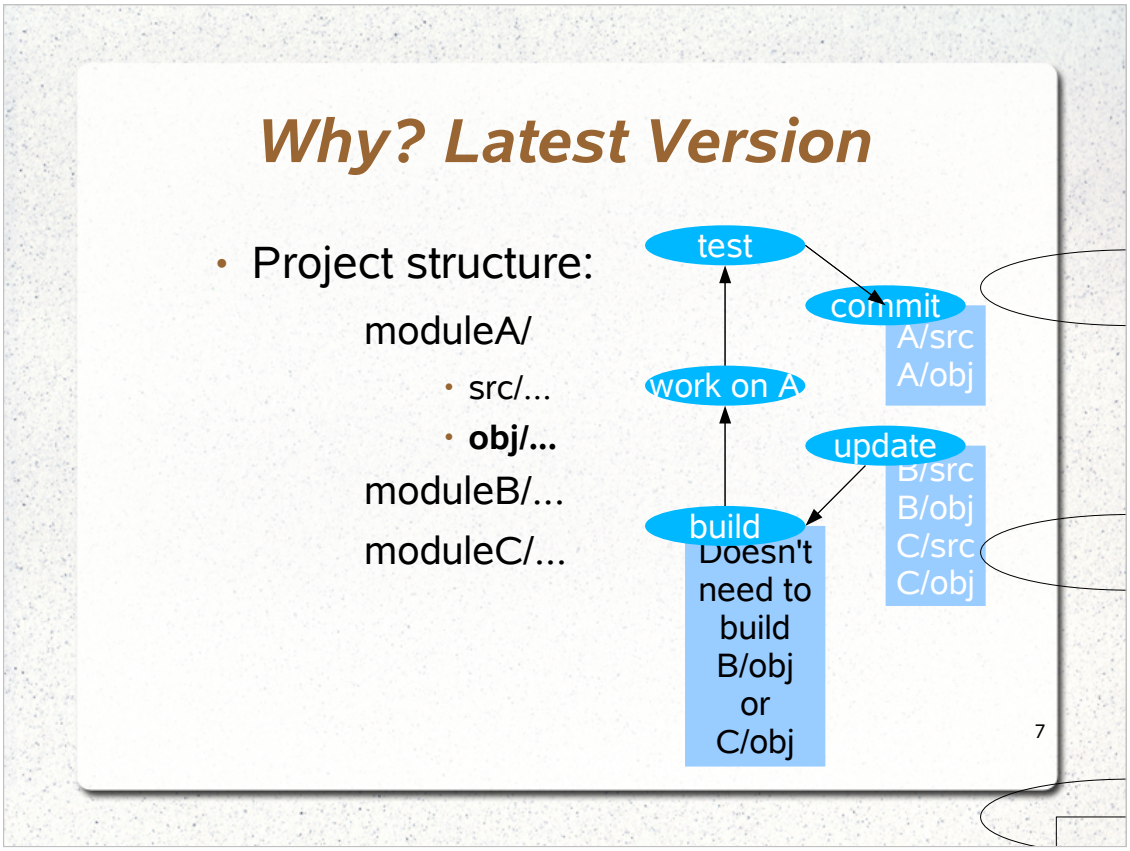


This is one of the most important use cases.
There has also been talk of wanting to remove sensitive data from server-side database files, but that seems to be a much less common requirement.

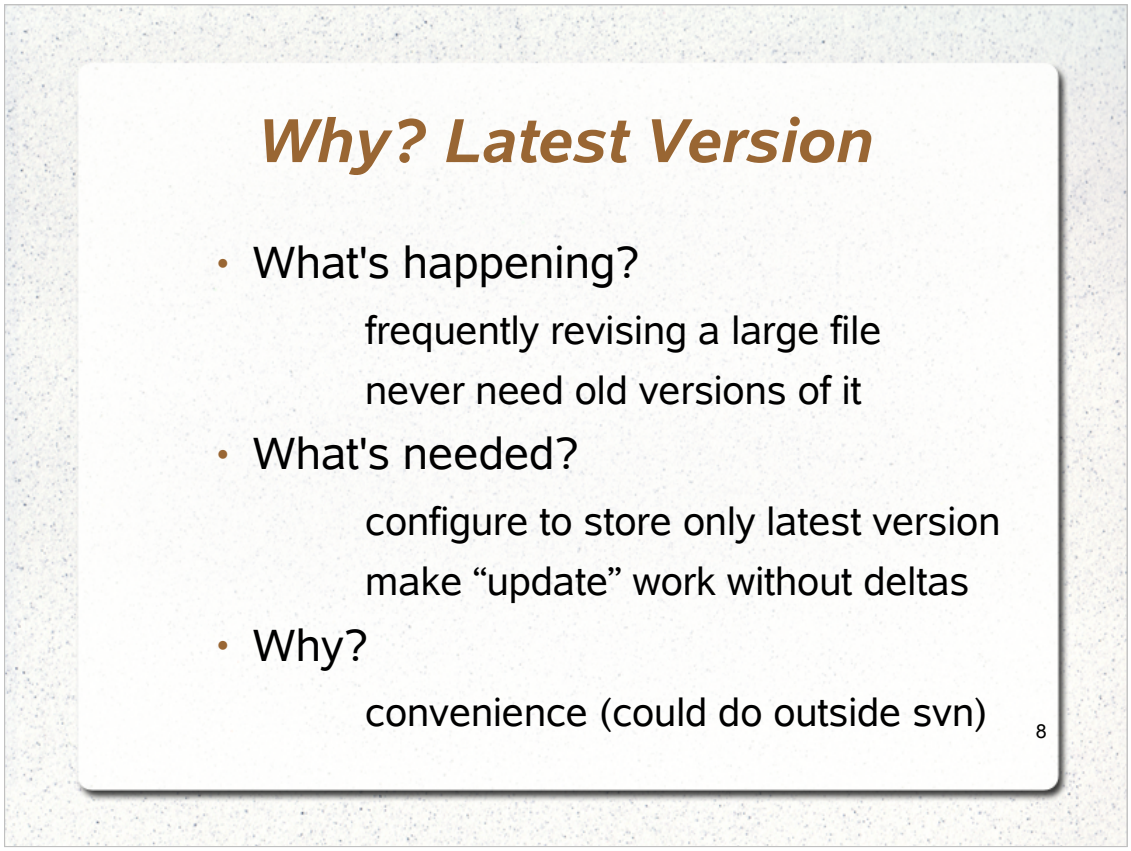


This is one of the most common and important use cases.

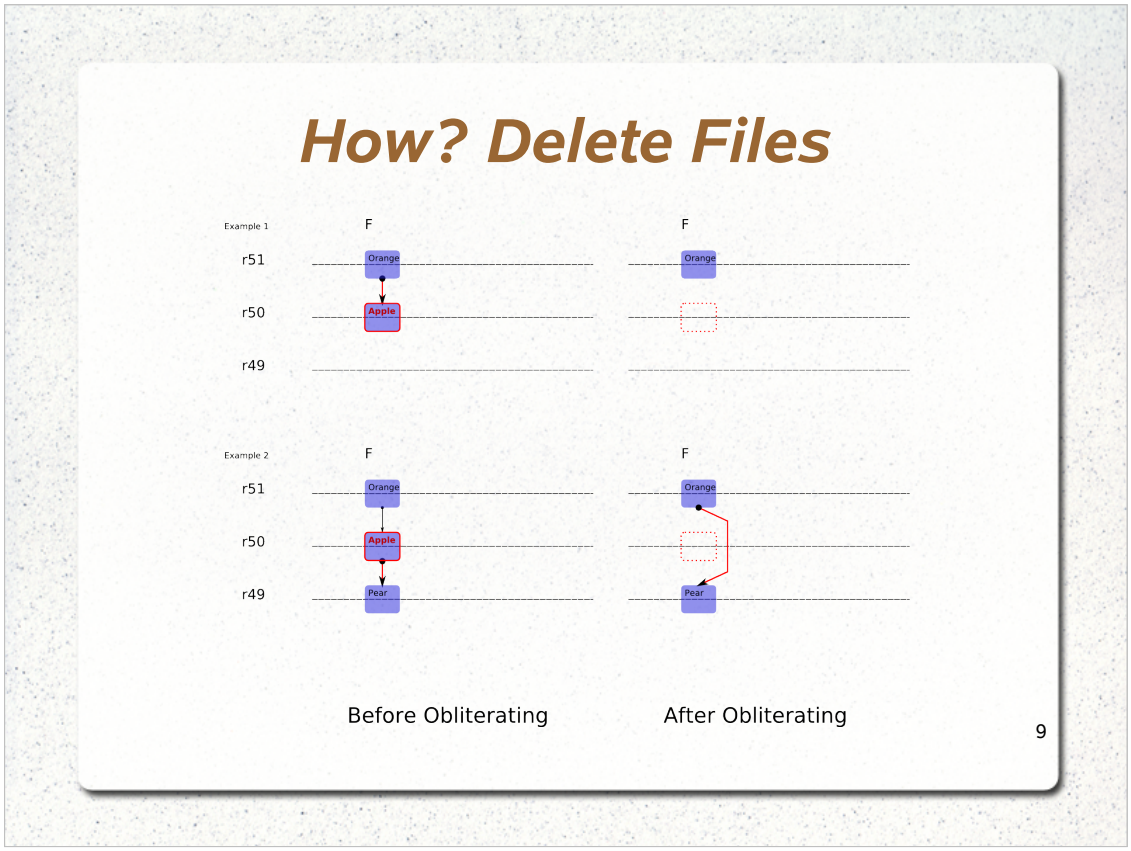
Even when correcting a mistake just made, can't assume that it's sufficient to delete just the most recent N revs of the whole repository, because the need arises most commonly in a busy repository.



In this example of a developer's work flow, we assume that the “obj/” directories are configured such that only the latest version is stored in Subversion. The developer working on module A is seen updating their working copy, receiving both source and object code of B and C. This means A doesn't need to build modules B and C before being able to continue working on A, thus saving time. In the rare instances when a developer needs to check out an older version, then the developer does need to build all the modules because the old versions of the “obj/” directories are not stored.



(A design note re. making “update” work without deltas. When Subversion updates a WC, it normally expects to transfer just the changes. Since the server in this case doesn't have a copy of the old version, we need to modify the “update” mechanism to cope with that. It's not necessarily difficult. We also need to consider how the behaviour of this kind of update (a nonexistent version to/from an existing version) relates to the behaviour you get if you ask to update a nonexistent version of a normal Subversion-controlled file to/from an existing version of it. There are important differences.)

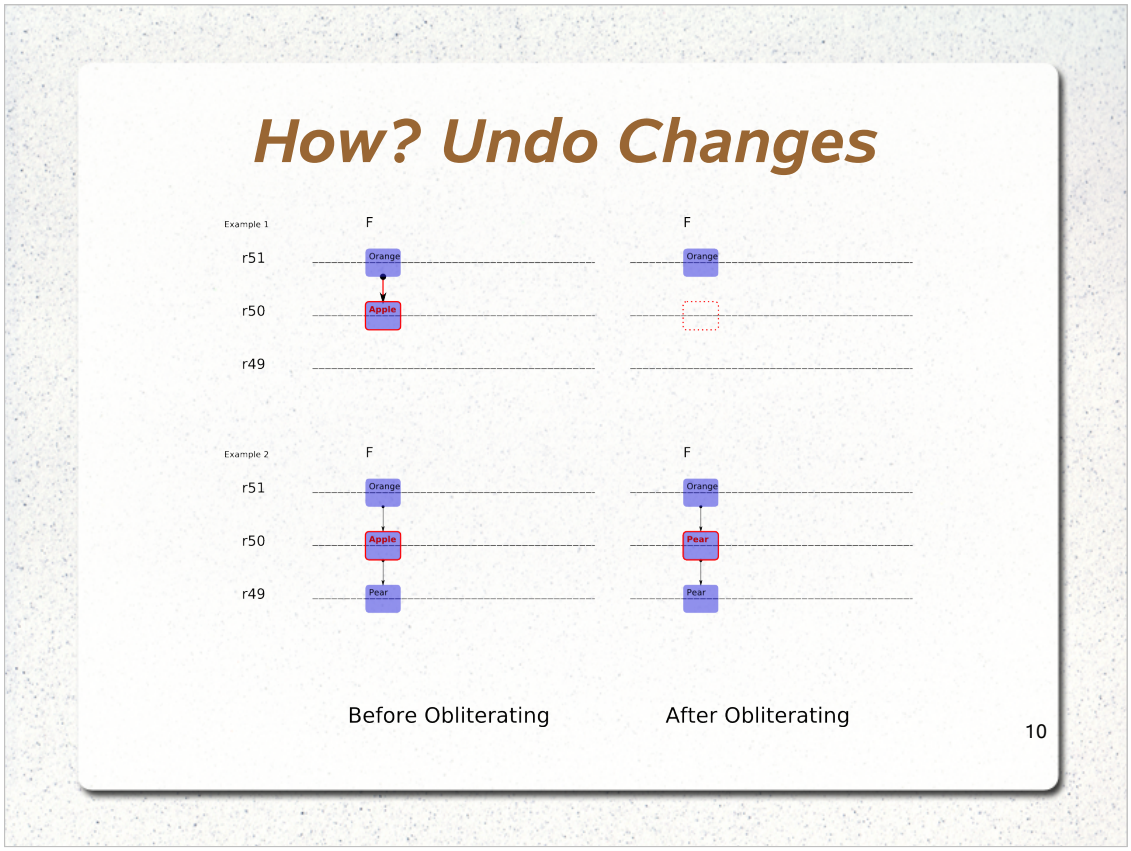


There are many possible ways in which history could be changed. This is one way, and on the next slide is another way. Both of these ways satisfy all of the use cases.

This scheme, here labelled “delete files”, is in fact illustrating deletion of an individual node-rev. The user can obliterate individual files (or directory trees) from individual revisions (or ranges of revisions).

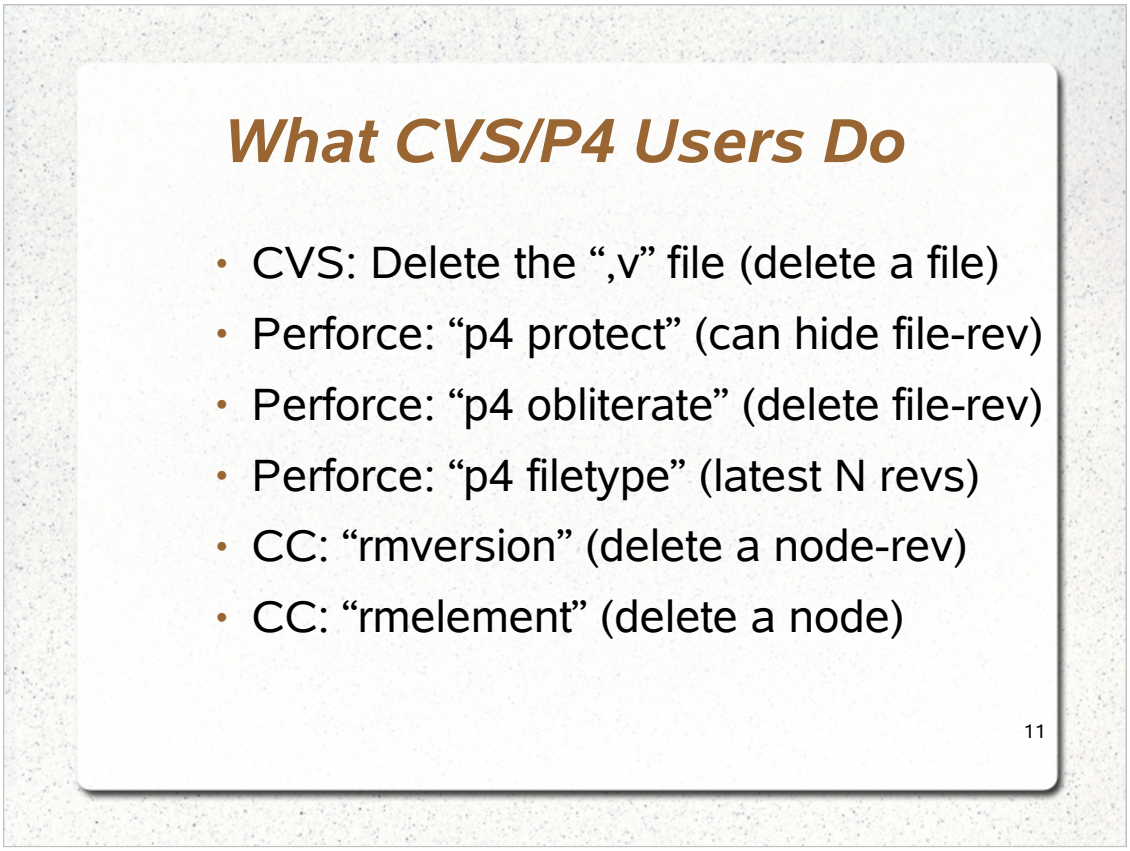
This scheme shows history being re-connected from F@51 to F@49, whereas another option is not to re-connect the history in this way.

See <notes/obliterate/fspec-dd1/> in the source tree for details of this scheme.



This scheme, here labelled “undo changes”, differs from the “delete files” scheme in that it replaces an individual node-rev with the previous revision of that node, or deletes it if it did not exist in the previous revision.

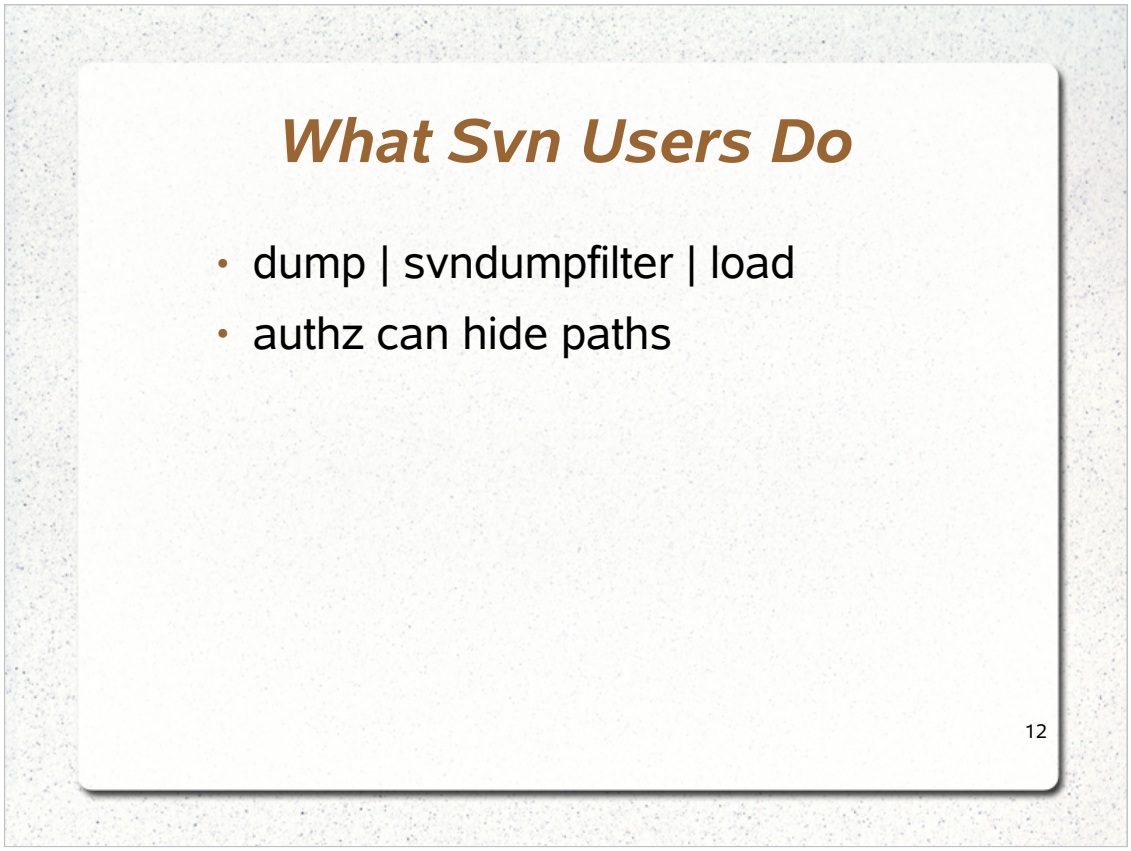
See <notes/obliterate/fspec-cc1/> in the source tree for details of this scheme.



A CVS admin can remove an archive ",v" file to effectively delete the whole history of a particular file. (Recall that in CVS each file has its own history, version numbers and log messages.) It is not so easy to remove selected revisions or changes from a CVS archive.

A Perforce admins can hide files with "p4 protect", delete file-revisions with "p4 obliterate", and enable "last N versions only" on selected files with a "p4 filetype" setting.

A ClearCase admins can remove a node-rev with "rmversion" or remove the whole history of a node with "rmelement".

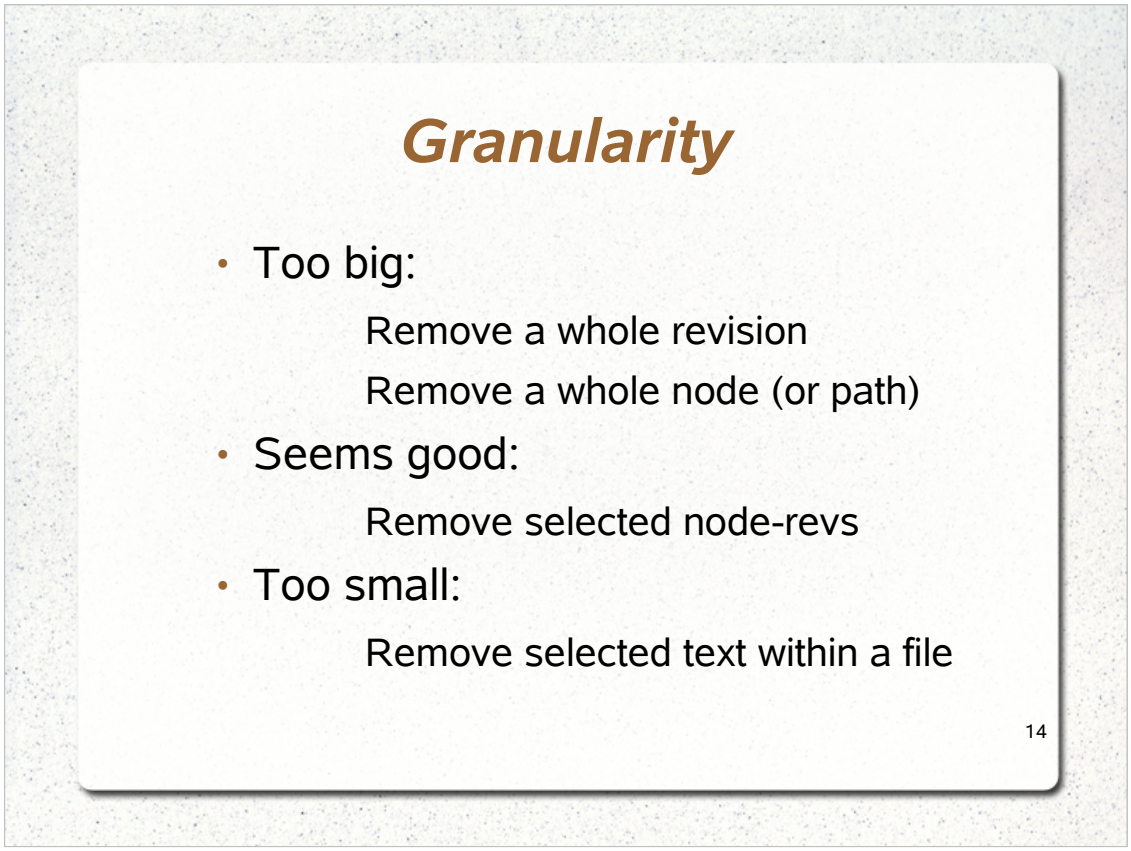


Subversion administrators can dump, filter and load a repository. Functionally, that satisfies most needs, but on a large repository it is much too slow for anything except a planned space-recovery.

Authorization settings can be used to hide specific paths. (But not at specific revs? Need to check.)

Non-reasons

- These possible reasons to obliterate are not goals
  - to undo (roll back) a recent commit
  - to tidy up history after an unwanted change has been done and undone
- Documentation should steer users to best practices for these tasks



All of these granularities have their uses, and have the same conceptual issues (such as how clients re-synchronize). We must pick a sensible granularity that satisfies normal use cases without over-engineering.

Any implementation with a small granularity can of course be used to perform larger-granularity tasks as well, e.g. if the granularity available is NODE@REV we could obliterate "/trunk/foo@{1:HEAD}" or "/@54321".