BESSY

Berliner Elektronenspeicherring-Gesellschaft
für Synchrotronstrahlung m.b.H.

Experience with a Distributed Revision Control System

Leibniz Gemeinschaft

Things we all know about CVS

Weaknesses

- cannot version directories
- thus cannot handle renames (while keeping history)
- has no idea about (atomic) changesets
- merging between branches is difficult and error-prone
- working copies contain no history
- (only) one central repository per project

Strengths

- very stable
- fast (enough)
- portable
- problems well-known, certain work-arounds exist
- GUIs (tkCVS, ...)

What do we do about it?

everyone wants to get away from CVS, but...

- there is no clear established successor
- instead: many different tools, some similar to CVS, some very different
- some of the more well-known alternatives are:

  Subversion
  SVK
  Arch(tla)/ArX
  Bazaar
  Darcs
  Git
  Mercurial
  Monotone
  ... (there are many more)

What do we chose to replace CVS?

What BESSY (control system group) did

- wide-spread dissatisfaction with CVS
- common conviction that it should be replaced
- plus rising pressure due to upgrade from EPICS 3.13 to 3.14
  (development on two separate branches in parallel; experience told us *not* to do this with CVS)
- but no effort to evaluate all the alternatives

  => choose the most conservative solution => *Subversion*

- rationale: minimize risk (stable product), maximize interoperability (e.g. support for integration with bug trackers like *trac*)
- initial effort to convert our repositories from CVS to Subversion was seamless and easy (though one-way)
- a promising start...

Subversion

- centralized model, like CVS
- but solves many of its shortcomings: versions directories, handles renames, atomic changesets, ...
- notions of *branch* and *tag* unified under the concept of history preserving *copy*

- the major problem: development on multiple branches in parallel remains very maintenance intensive
  - Subversion does not track which changes have been merged from one branch to another (external tools exist which support this; still tedious)
  - if a change gets merged twice, many spurious conflicts result which have to be resolved manually

  => merging is difficult and error-prone

Darcs: Overview

- fundamental notion is *change*, rather than version
- based on a mathematical formalism (*theory of patches*)
- takes the idea of decentralization to its extreme
- unique merging capabilities
- interactive command line interface
- written by a physicist (David Roundy), programmed in Haskell :-)

Darcs: Change based, Patch Formalism

- fundamental notion is the *change*
- a version is a *set* of changes (applied to the empty source tree)
- a *patch* is a description of a change
- a set of changes is stored as a *sequence* of patches

- primitive patches include
  - *hunk* (zero or more adjacent lines in a file replaced by other lines)
- - *rename*, *move*, *add* and *delete* files and directories
- - *replace a token* (unique darcs feature)

- patches have certain algebraic properties
- they are all *invertible*
- two patches can *commute* => they can be applied in any order
- if not, then one *depends* on the other
- darcs discovers dependencies and selects dependent patches if necessary

Darcs: Radically Decentralized

- strictly 'egalitarian': all repositories are (technically) equals
- working copy == repository == branch
- select patches from anywhere in the history of a repo (*cherry-picking)*
- directly exchange patches between repos
  => no 'central repository' bottleneck

- *history* is a local concept => no global history for the whole project
- knowledge about whole project is indeed *distributed* among existing repos

- remote access via standard protocols (ssh/http/mail), no special protocol or server needed, read only access (via http) is trivial to administer

Darcs: Branching and Merging

- patches are identified by *timestamp*, *author* (email address), and *name* (one-line comment)
- patches are globally unique entities
- usually selected by name

- merging is a day-to-day activity
  - no merge command: 'push', 'pull', and 'apply' all automatically perform merging when and if needed
  - easy to avoid conflicts
- in case of conflict:
  - darcs marks conflicting patch as a special *merger-patch*
  - conflicts have to be resolved manually

Darcs: User Interface

- very nice, interactive command line interface
- no GUI yet :-(
- some nomenclature

`record`	locally create a patch (like *cvs commit* but off-line)
`pull`	receive new patches from a remote repo
`push`	submit new local patches to a remote repo
`send`	create a patch bundle and send per email to author
`apply`	apply a patch bundle (e.g. received per mail)

Experiences using Darcs

- easy to learn and use
- branching and merging is simple, safe, and effective
- often used: locally record changesets w/o publishing them, e.g. for
  - temporary debugging code
  - experimental changes
- often used: local branches

- control system development
  => upgrades must be incremental
  => multiple branches with many parallel changes

- BESSY internal work flow not much different
  - central repository contains the 'official head' of both the EPICS 3.13 and 3.14 branches (for each project/module)
  - developers keep local branches/repos/working-copies as they see fit
  - developers push their changes to central repo (after testing)

Experiences using Darcs: Caveats

- recording changes separate from publishing them
  => need to remember to publish changes
- tags are different
  - a *tag* is a *null change* which (artificially) depends on other patches
    (by default those that exist in the current repo at the time the tag is created)
  - simplify reproduction of a certain *version* of the source tree
  - regular tagging is good
  - need convention for tag names (so they are unique)
- keep patches small and independent
  => avoids conflicts
  => think before recording
- unusual: darcs does *not* preserve/track file permissions
- security agnostic: except support for ssh, access must be restricted by underlying OS / filesystem

Experiences using Darcs: Distributed Development

- easy to give world-wide read-only access via http
- sending patches is extremely light-weight
- greatly reduces the entry barrier to contribution

- if you don't (yet?) want to contribute
  - locally recording changes insulates against upgrades
  - easily removed or re-added
  - share patches with collaborators, bypassing main development trunk

- EPICS development (both core and support modules) could greatly benefit from such an RCS: more contributions, less maintenance

The Darc Side of Darcs

conversion from/to other RCS could be better

- available tools: cvs2darcs (perl script) and tailor (python)
- have seen situations where both both have problems to correctly convert a CVS repo w/o manual intervention

still has a number of serious bugs

- known situations where darcs crashes and leaves the repository in a bad state
- we recently encountered one of these (found a way to fix our repos, but tedious)
- one safe-guard is to record pending changes into a dummy patch prior to pulling e.g. from the central repo

The Darc Side of Darcs

sometimes 'hangs forever'

- merge algorithm in certain cases exhibits exponential blowup
  => extremely long running times (hours and days)
- currently being worked on with high priority
- circumstances:
  - large patches with many conflicts
  - particularly so-called *doppelgaenger-patches*
- avoid by keeping patches small
  - also reduces likely-hood of conflicts
  - makes later cherry-picking easier

BESSY

Berliner Elektronenspeicherring-Gesellschaft
für Synchrotronstrahlung m.b.H.

Thank you for listening!

Leibniz Gemeinschaft