Berlin '13 hackathon will be held at the elego offices in Berlin, Germany from 10th to 14th June. elego has generously donated office space and $BEVER for the duration of the week, and several committers will be on hand to hack, discuss, and make themselves merry.

Potential Items for Discussion

- Goals / ideas for Subversion 1.9
  - Are we aiming for a 9 month or rather a 18 month project?
  - Process changes so we can avoid incomplete features from blocking a release?
  - Non-binding improvement / feature wish list
- Merge
  - Do we need a new data model? How would that look like?
  - What needs to be done for move support?
    - Implement Move Tracking – see MoveDev How to Add Moves to Svn
  - In what aspects can the current merge algorithms / infrastructure be improved besides adding move support? (What problems are users still experiencing unrelated to moves?)
  - Refactoring our 500+kB merge.c?
    - One issue: Updating mergeinfo (props) in the WC is currently done two ways – directly, and via a 'result_mergeinfo' output hash that is written to disk later – but neither way is fully supported by all the code. Need to unify.
- Branch cleanup
  - Which of the existing branches are obsolete and can be removed?
- FSFS format 7
  - Goals and feature overview is given as a WebEx talk on Wednesday fsfs-format7.pdf
  - Technical discussion needs to be separate from that
  - People's feature wish list
  - How to get that into /trunk? This includes organizing a review of the refactoring and improvements for f6. f7 features can be disabled (similar to what we do with Ev2 right now).
- Shared repository cache
  - A.k.a. "cache server". What it is and why it's useful.
- Coding Standards
  - GCC recently allowed C++ for its own code. What is the general opinion amongst SVN devs toways doing the same?
  - Parameter checking to avoid NULL values causing security bugs?
- Pipelining the client
  - This seems to be the only available option to make the client scale better for large projects. Is that approach feasible for a 1.10+ timeframe and what would it take?
- Benchmarking
  - How could a systematic scalability and performance test for SVN look like?
- Tests
  - Should coverage reports be part of our CI builds?
  - Do we want to improve coverage? If so, where and when?
  - Ways to improve our test suite (started with a quick brainstorming by JulianF and DanielsSh at pre-hackathon dinner):
    - Knobs: Test with various combinations of sharding, pack-after-commit, server_minor_version, and all the other knobs.
    - Non-repo-root checkouts.
    - Filenames: Use spaces (and other awkward chars) in (WC, repo-root, versioned-FS) filenames.
    - Large data sets: Use large data sets as a matter of course – both simple ones to check basic scalability, and realistic ones to discover complex interactions. For example, design many of the simple tests to be able to run on top of an already large and complex repos/WC state. (And also able to run on a simple, empty state, to make debugging easier.)
    - WC states: Instead of the simple Greek tree, use a basic test starting point that contains combinations of (added, del, replaced, moved, mod) x (depth=N) x (...).
    - Client/server version combinations.
    - Include an svn:ispecial file that's not a "link", for future regression testing.
    - Unit test framework: Use Python bindings for C APIs so we can share test infrastructure. (Or provide Python bindings to libsvn_test and use that as the framework.
    - How to encourage (or to avoid discouraging) test-focused people from outside the project?
    - Should we split our tests in a fast "basic" and an slower "full" test set? (The former should be run before committing, and the latter is run on the build bots, and before a release.)
- Bindings
  - C++HL leading to a consistent binding interface between languages?
  - JavaHL native implementation eventually replaced by C++HL?
- Community
  - How can we attract more contributors?
  - How can we more thoroughly collect and understand user expectations?
- User work flow improvement
  - Ways we can make users lives easier (auto pager, stash, built-in bisect, interactive commit, patch submission automation).
- Versioned Object Model Improvements (Blue-Sky)
  - Metadata Indexing (have slides about directory index --brane)
  - Almost-first-class branches
  - Tracking and non-tracking links
  - see metadata-tng.pdf
- User-requested issue fixes for 1.9:
  - http://subversion.tigris.org/issues/show_bug.cgi?id=1256 (mtime preservation)
  - http://subversion.tigris.org/issues/show_bug.cgi?id=3457 (replication of locks with svnsync)
  - http://subversion.tigris.org/issues/show_bug.cgi?id=3750 (hotcopy of locks)
  - http://subversion.tigris.org/issues/show_bug.cgi?id=2662 (authz with wildcards)
Discussion Notes

The following are notes taken during developer discussions on the above topics at the Hackathon. These are non-exhaustive and likely not wordsmithed for public consumption.

FS-NG

brane presented his ideas for a revamped filesystem logical implementation that theoretically solves many of our common-most filesystem woes:

- metadata-tng.pdf

Release Cycles

In general, attendees expressed interest in shorter, time-based release cycles. The sole reason stated for feature-driven releases was for the purpose of always showing demonstrable progress on our users' needs (for example, merge tracking), but we generally agreed that this is primarily a communications problem.

Much discussion occurred around the mechanics of branch-based feature development, the possible introduction of APIs marked "experimental", and the ever-present necessity of getting eyeballs on features early while still wishing to keep the trunk trending toward relative stability as the end of the time-based release cycle approaches.

The following is the joint recommendation of the hackathon attendees:

In the interest of serving our user base, we are proposing that each release live on the trunk for at most nine months. The first six months of this period are open to new features. In the first three months of the new feature period, large, destabilizing features will be accepted (under the provision that the feature itself is arguably "complete"). In the second three months, we will accept smaller new features. But at the end of the six months, the trunk is closed to new features and stabilization for the release begins. We will allow up to three months for stabilization on the trunk, cutting the release stabilization branch at the end of the period (or beforehand, if we agree that the codebase is ready for RC). After the point, the release branch is managed the same as it is today, with an RC at the earliest reasonable moment and the same soak requirements and such that we've always had.

Some open questions remain, namely:

What affect should this have on our support of our release lines? There was some support for the idea of moving to time-based support cycles rather than our current approach of maintaining the most recent major release only. Shorter release cycles should lead to (smaller-impact) releases more often, but adopters will realistically not absorb every single Subversion release. It might be beneficial to say that we'll continue to maintain 1.X releases for a period of, say, two years from their release date. (We can, of course, choose to patch even older releases for security or other high-impact fixes, of course – it's only our minimal promises that we're talking about here.)

Should we require vote-based approval on the reintegrations of feature branches? At least some of the hackathon attendees favor this (with the typical "three +1's and no vetos", specifically). This both helps to solve the problem of code bombs (that is, minimizes the cognitive destabilization) and also encourages feature composers to do a better job of vetting their designs in advance so as to a) ignite interest and attention and b) reduce the change of widespread disapproval of the feature or the approach taken.

FSFSv7 Branch Reintegration

stefan2 expressed that while he is confident that FSFSv7 is solid code, it's also quite critical and could easily take a year or more to fully stabilize. Attendees felt that perhaps it would be best to introduce FSFSv7 as a new, experimental fs-type. Stefan said he had been thinking about the same thing himself, even considering a different name for his implementation. Some discussion was had around how to manage the common code found in the two implementations. Stefan will explore the feasibility of a storage abstraction layer here, but others seemed fine with the code duplication, if any because it would provide an opportunity to purge the code specific to older FSFS formats from the new fs-type.

Shared repository cache

Tentative agreement with the idea (with admission that stefan2 has clearly thought more about this than anyone else). Concerns here were around security of the shared memory. Discussion deferred.

Parameter checking to avoid NULL values causing security bugs?

Discussion began around static analysis tools and runtime checking. A sub-thread began around the possibility of ditching the mod_dav + mod_dav_svn pairing, and introduce a new module that covers the combined domains of both (but which is under the control of this project alone). As a first step, we need to protect the surfaces exposed to the network. Approval was expressed for breser's specific ideas around tooling (based on build.conf) to drive clang in the most beneficial way possible.

Pipelining the client

stefan2 was polling the attendees regarding the feasibility of introducing multi-threaded handling and pipelining into the Subversion client layer. While today Subversion is largely I/O-bound, the concern is that at some point, Subversion will instead be CPU-found. How will we progress past that point?

User acceptance testing

CollabNet, WANdisco, elego need to be taking advantage of customer connections earlier and more often. While users@ has traditionally been a support forum, we are missing the opportunity to get UAT of even not-yet-coded features by not contacting users@ with our ideas and plans. We talked about trying to ensure that our next issue tracker have a workflow engine that would remind us to do post-dev UAT, too.
Attracting developers

stefan2 noted that, going forward, we can probably expect that our client user base will be increasingly more Windows-focused. Unfortunately, Windows is the platform on which building Subversion is hellishly complicated. Animated discussion followed about how to make it easier to build on Windows. Consensus was, to say the least, not achieved.

But what about other developers? jcorvel points to the simplicity of git users to clone a project repository and get to work. Perhaps we should provide some kind of bootstrapping script for creating a local repository, import Subversion's code, make a working copy of the Subversion source code, etc.

(danielsh: I've started http://subversion.apache.org/quick-start and http://subversion.apache.org/docs/community-guide/general#patches-writing in response to this item. Feel free to iterate, improve, etc.)

C++HL leading to a consistent binding interface between languages?

Attendees seemed to agree in general to this idea.

We talked a bit about bindings compatibility promises. Most of us felt that the only bindings which see large-scale consumption are the JavaHL bindings. Realistically – and due to developer neglect – there is a break in compatibility every time we notice (too late) that some new interface lacked proper "swiggification", so we didn't feel like compatibility in the Swig-built bindings was so very important.

JavaHL native implementation eventually replaced by C++HL?

Sounds like a good idea, and we don't anticipate that deciding to do so would cause the C++ wrapper to be unnecessarily constrained by the need to maintain compatibility in the existing JavaHL API.