Version Control API: An introduction

Jakob Petsovits, DrupalCon DC, March '09

Why is it necessary at all?

On the one hand, because version control is cool, and by integrating it with Drupal it's possible to let the CMS react on changes in the repository and add stuff to nodes. Also, Drupal themes look better!

On the other hand, drupal.org uses cvs.module (a.k.a. cvslog) to handle many of its release management features, like tarball packaging, CVS account management and commit restrictions, displaying CVS messages on http://drupal.org/cvs, etc. - as long as d.o depends on cvs.module, we can't even consider switching core and/or contrib to another version control system (VCS).

Objectives

- No assumptions towards a specific VCS. (Tailoring towards specific feature sets such as “distributed”, “supports directory revisions”, etc., is ok.) Code supporting one VCS should support all of them.
- Modularize functionality so that sites only need to use the features that they actually need. Should work just as well for sites other than drupal.org.
- More possibilities through a generic API instead of hardcoded workflows and lots of db queries.
- Get rid of cvs.module on drupal.org.
- Make VCS integration with Drupal rock!

Timeline

- 2007: created as GSoC project by jpetso, most functionality ported from cvs.module, 5.x-1.x times
- 2008: Git/Hg backends through GHOP, SVN backend through university, major refactoring, follow-up GSoC project failure, module nearly dies, sdboyer takes maintainership. Module recovers, refactoring gets done, 5.x-2.x is stabilized
- Jan-Mar 2009: port to Drupal 6, implementing cvs.module's last missing features, 6.x-1.x
- < July 2009: deployment on d.o
- > July 2009: jpetso leaves → developers needed

The big idea: three layers of responsibility

Version Control API specifies a set of mandatory and optional functions that can be implemented for every common VCS. Backends implement those functions as good as possible. Higher-level modules use those functions for integration purposes.

Central concepts and ~~objects~~ array structures

On the API side, Version Control API revolves around the same concepts present in every common VCS:

- Repositories: local or remote, one or many. (With DVCS like Git, their number is potentially huge.) Most admin preferences are stored per repository.
- “Accounts”: just a (probably incomplete) association of Drupal uids to VCS usernames.
- Items (a.k.a. item revisions): files or directories that belong to a repository, including path and revision. Most item revisions, but probably not all of them, are recorded in the database.
- Labels: the unified term for “branches and tags”. Also stored in the db, but potentially unreliable as they can change without us knowing.
- Operations: stuff that happened in a repository at a specific time. That includes commits as well as the creation and deletion of branches and tags (which is then called “branch operation” or “tag operation”). An operation includes information about revision author, revision number/id and the log message, and is associated to any number of items and labels. In short, operations are what you see on http://drupal.org/cvs.

Version Control API is based on the idea that the current state of a repository is essentially unknown, but all log information up to a certain point in time is available in complete form in the database so that commit logs can be shown (and commit statistics calculated) without invoking the VCS binary.

For browsing the repository, direct interfacing with the VCS itself is required. Also, the association of items to branches and tags cannot possibly be recorded in a correct & maintainable way, so determining that is also left to on-the-fly invocations.

Version Control API takes care of managing the above entities, and provides hooks for modules to act when e.g. a commit has been recorded.

API user's instant favorites

- hook_versioncontrol_operation(): invoked when an operation (commit, etc.) is recorded in the db.
- hook_versioncontrol.php: extensive API docs for all the important hooks.
- versioncontrol.module: comes with huge lots of API docs (better run it through doxygen, maybe).
- FakeVCS backend: API docs for backend writers, with examples on how functions might look like.

As a user, what can I do with this right now?

- Display commit messages with the Commit Log module, also available per project (optional) and containing links to external repository viewers (optional, too). The same is also available as notification mails and RSS feed.
- Export accounts to a CVS passwd file, making account management through a Drupal UI possible. Optionally with admin moderation for account registrations. Unfortunately, the account architecture is b0rked so this won't be possible with SVN/Git/Hg/bzr unless it's being reworked.
- Disallow commits for users if they try to create an incorrectly named branch or tag, or if they are not listed as project maintainers of a project node.
- Rules integration: Let the Rules module fire any events when a commit is recorded. Say, mail a notification mail to users in only a certain group only if it happened in a specific directory.
- New new new! Generate release tarballs using the versioncontrol_release module. (Still in active development, but basically works.)

What Version Control API will never do:

Everything that involves working copies. From the beginning, Version Control API was conceived only for server-side tasks so that merge conflicts, and other difficulties stemming from a modified working copy, can be avoided. In short, you won't be able to perform automated commits with this module.

Plans for the near future and medium term

- Next up: get everything ready for d.o deployment.
- Next up: drastic speed-up of the SVN log parser (which is necessary because it's dog slow).
- Multiple accounts per user and repository.
- Sane user authentication architecture, so that it's possible to export accounts to common authentication methods like .htaccess or SSH keys (and let the user select which one to use).
- Refactor the API in an object-oriented fashion, for better flexibility and performance improvements like lazy loading of objects.

Get involved!

- Help out with the above, look out for further ideas at at http://groups.drupal.org/node/19469, or simply hack on whatever stuff you'd like to see.
- http://drupal.org/project/issues/versioncontrol and other issue queues for concrete action items.
- http://groups.drupal.org/revision-control-systems for general brainstorming about VCS topics.

Hands-on: A minimal setup with SVN

Step 1: create an SVN repository. (Skip this if you've already got one.)

- `svnadmin create /home/joe/testrepo`
- `svn co file:///home/joe/testrepo ↵ /home/joe/testrepo-checkout`
- `cd /home/joe/testrepo-checkout`
- `echo "Blah" > a.txt && svn add a.txt`
- `svn commit -m 'Initial Blah!'`

Step 1, alternative version: create a CVS repository.

- `cvs -d /home/joe/testrepo init`
- `cvs -d /home/joe/testrepo co ↵ -d /home/joe/testrepo-checkout .`
- `cd /home/joe/testrepo-checkout`
- `echo "Blah" > a.txt && cvs add a.txt`
- `cvs commit -m 'Initial Blah!'`

Step 2: set up your Drupal site with that repo.

- Enable the Version Control API, Commit Log and Subversion (or CVS) backend modules.
- Go to Project administration → VCS repositories → Add Subversion repository in the admin area. (Or “Add CVS repository” if you did that one.)
- Repository name: “Test repo”, or whatever.
- Repository root: “file:///home/joe/testrepo” (SVN) or “/home/joe/testrepo” (CVS).
- Save the repo, then follow the “fetch now” link.
- Enable the “Commit messages” menu entry, or go directly to the “commitlog” path.
- The commit shows up there (and the message log will be updated on every cron run). Win!

Hands-on: Hooking up the CVS hook scripts

- cd `/home/joe/testrepo/CVSROOT`
- `chmod u+w *`
- Make sure `UseNewInfoFmtStrings=yes` exists / is uncommented in the “config” file.
- `cp $DRUPAL_ROOT/sites/all/modules/ ↵ versioncontrol_cvs/xcvs/*.* .`
- `chmod a+x xcvs-*`
- Adapt xcvs-config.php as described in there.
- Put the lines listed in README.txt into the “commitinfo”, “loginfo”, “taginfo” and “posttag”.
- Go to Project administration → VCS repositories → (edit the CVS repository), and switch “Update method” to “Use external script to insert data”.
- Commit something.
- Shows up on the “Commit messages” page!
- Bonus feature: enable the “Commit restrictions” module, and define a set of allowed branches and tags – you can't commit into those now.