The New System for Releasing Drupal Contributions

Derek Wright <drupal@dwwright.net> http://drupal.org/user/46549

Problems of the current system

Today, Drupal contributions are numbered “`X.Y.0`”, where “`X.Y`” correspond to a given version of Drupal’s core API. There can be only one branch of the module compatible with a given version of core. As developers fix bugs, new “releases” [sic] are automatically produced with the same number and name (only the timestamp on the filename changes). If a module author wishes to keep the official “releases” for a given core API stable and only fix bugs on that branch, their only option for new features is to use the trunk of the CVS repository, which forces people to use an ambiguous `cvs` version of their module, which could be intended for *any* version of Drupal’s core API. Bug reports and documentation refer to completely ambiguous names that cannot be reliably recreated, since there are no CVS tags to recover a specific version number.

Goals of the new system

We need official releases for all Drupal contributions, not just the core system. These releases must be intentional, tagged in the CVS repository, and uniquely identified. There should be no ambiguity when referring to the code running on a site, in issues, bugs, tasks, security alerts, documentation, forum posts, or in emails.

The other primary goal is to have multiple branches of development that are compatible with the same version of Drupal’s core. This will allow separate stable and development versions of contributed modules, allowing developers to add new features without endangering the stability of the code, which can be immediately deployed on sites running a given stable version of core Drupal.

Additionally, the new system will ensure that no one ever has to run a release from the TRUNK of the Drupal CVS repository, unless they plan to develop and test the very latest code that’s being ported to the next version of the Drupal core API. Tremendous confusion exists because the TRUNK (sometimes called “HEAD” or just “CVS”) is a moving target, and the state of the code for a given contributed module in the TRUNK is usually unknown.

Constraints

Albert Einstein wisely said, “*Make everything as simple as possible, but not simpler.*” Unfortunately, Drupal development happens in a very complicated context, with rapidly changing versions of the core API, the interaction of different modules, the need for stable and development versions of many modules, and so on. Balancing the needs of a simple, usable system with something powerful enough to solve the problems we currently face has been a constant challenge. The other constraint is that the project.module and other related parts of this system (project_issue.module, cvs.module, etc.) must be usable on sites other than drupal.org.

Version numbers

Core version numbers would remain the same: `Major.Minor.Patch.` However, contributed modules would now have numbers of the form: `SuperMajor.SuperMinor-Minor.Patch`, where the `SuperMajor.SuperMinor` portion would refer to the version of the core API the release was compatible with

CVS branches

Core branches would continue to be named as they always have: `DRUPAL-X-Y`. The default stable branch of any contributed module would be the same, just like now. However, instead of releases from this branch being called `X.Y.0`, the first release would be `X.Y-0.0`. For example, the first official stable release of a module for the Drupal 4.7 API would be version `4.7-0.0`. Subsequent stable releases would increment the module’s patch level, becoming version `4.7-0.1` and so on…

If a project maintainer wishes to provide a development branch for a given version of the core API, they would add a branch of the form: `DRUPAL-SuperMaj-SuperMin_Minor`
For example, the first development branch for the 4.7 API would be `DRUPAL-4-7_1`. Mixing – and _ in the tag name could potentially lead to confusion, but CVS tags have a limited set of allowed characters, and I believe we need a visual separation between the part of the tag referring to core and the rest. Calling the above tag `DRUPAL-4-7-1` would look exactly like the release tag for the `4.7.1` version of Drupal core. This is unacceptable.

What's wrong with "`DRUPAL-4-7-DEVEL`"?
Some modules will require multiple stable and development versions for the same version of core, especially modules that provide an API which other modules depend on. Furthermore, branch names should match version numbers as closely as possible (for simplicity I'm making an exception for the primary stable branch).

CVS release tags
The exact versions of every file that comprise an official release of the core system would continue to be tagged in the CVS repository using the existing convention: `DRUPAL-X-Y-Z`. Similarly, every official release of a contribution would have a CVS tag that identified it: `DRUPAL-SupMaj-SupMin_Minor-Patch`. For example, the `4.7-1.3` release of a module would be tagged `DRUPAL-4-7_1-3`.

Release workflow
Given all of this, how would a developer create a new release?
1. Write code or apply patches, and commit the changes to the appropriate CVS branch.
2. Once a set of coherent modifications are complete and a new release should be identified, the author creates the release tag.
3. They go to their project home page and create a release node. The developer selects the CVS tag to use (which determines the version string), and must provide a description of the release. At this point, the release node would be submitted but unpublished.
4. A packaging script runs automatically, queries the database for all unpublished release nodes, checks out a copy of the project directory using the specified release tag, creates the `[project]-[version].tar.gz` package, updates the release node to point to this file, and publishes the release node.
5. The project node automatically displays the latest published release nodes for each active branch, so the new release becomes visible.

Nightly development snapshots
Projects will continue to provide automated nightly development snapshots to aid end users that want to help test the next release. Every branch on a project, including the CVS TRUNK, can have its own release node. If a release points to a branch, not a tag, the packages are identified as "dev" releases and the version number is computed automatically.

Branch and tag validation
Enforcing these naming conventions for branches and tags is trivial. The access control scripts that run on drupal.org can now tell the difference between branches and tags, and use regular expressions to validate the input. With some additional work, the scripts could also ensure release tags are sequential on every branch.

What if I don't want to do any more work?
The degenerate case of this new system, where the author does no additional work over their current development habits, is nearly identical to the system we have now. If they only branch their code once they port to a new version of Drupal's core, they'll still get automatically-generated nightly snapshots from the end of that branch, the versions will continue to be ambiguous (e.g. 4.7-0.0-dev), and users will continue to have to keep track of the dates associated with a given snapshot.

Features we can build on this foundation
1. Email subscriptions and RSS feeds for new releases from an individual project (optionally filtered by branch)
2. New page for the *version compatibility grid* – a table of project names down the side, where each column represents an active version of Drupal core, and each entry in the table indicates if the module is compatible, links to releases for stable and development branches. (http://drupal.org/node/63491)
3. Extensions to release nodes that are specific to drupal.org (probably taxonomies)
- Database compatibility (MySQL, PGSQL, etc)
- PHP version
4. Adding an optional link on release nodes that points to the security announcement the release is the fix for (existence of the link could be used in various module listings to add some kind of "security fix" icon)
5. Recent releases block
6. Per-branch CVS access to project source code for individual users

Questions for the audience
1. Should the primary stable branch be version `X.Y-0.Z` or `X.Y-1.Z`?
2. For 5.0.X releases, should the packaging script automatically write the version string into the `.info` file when it makes a `.tar.gz` file?