

How to Effectively Manage and Release Your Drupal Contributions

Derek Wright (dww), DrupalCon Boston, March 2008

Why care about release management?

- Drupal 6 core contains update status notifications so sites know when to upgrade.
- Proposal for a killer feature in Drupal 7: "Automatic"(!?!) upgrades.
- If you manage your releases correctly, it will save you time and hassle.
- It allows people to actually use your code.
- It allows you to handle security problems quickly and safely.
- It makes it easy for you to keep moving your code forward and improving it without making life hell for your users.

Underlying assumptions

- No one is forcing you to do any extra work or to do anything -- you always scratch your own itch (or the itches of people paying you).
- No one requires that you put your code on drupal.org and share it with the world.
- The Drupal project thrives because so many people contribute their code.

Once you upload your code, then you have some responsibilities

Putting code on drupal.org implies that you think people should use it. Therefore:

- You have to be conscious of security vulnerabilities and be willing to fix them.
- You should clearly state your intentions and plans as a maintainer so users can prepare.
- You should be aware of your user-base.
- Poorly maintained code contributed to drupal.org gives Drupal itself a bad name.

Tools for maintainers

There are three primary tools that help you:

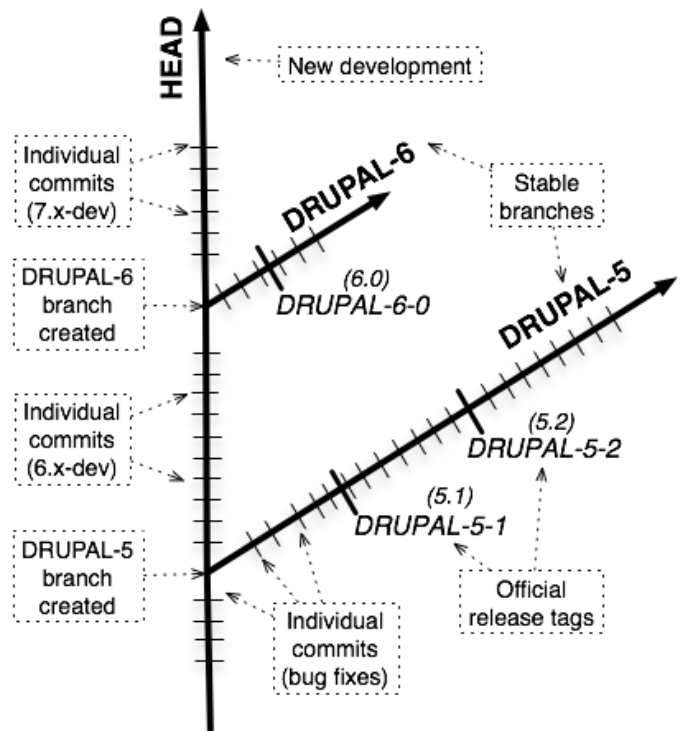
- A revision control system (currently CVS) lets you keep track of changes and is required to host your code on drupal.org.
- Creating releases (both official releases of a specific set of code that never changes, or development snapshots that are rebuilt automatically). Official releases are best.
- Project nodes on drupal.org allow you to describe your contribution, show what versions are recommended to use, and state your intentions as a maintainer.

Basics of revision control

There are three key concepts to grasp:

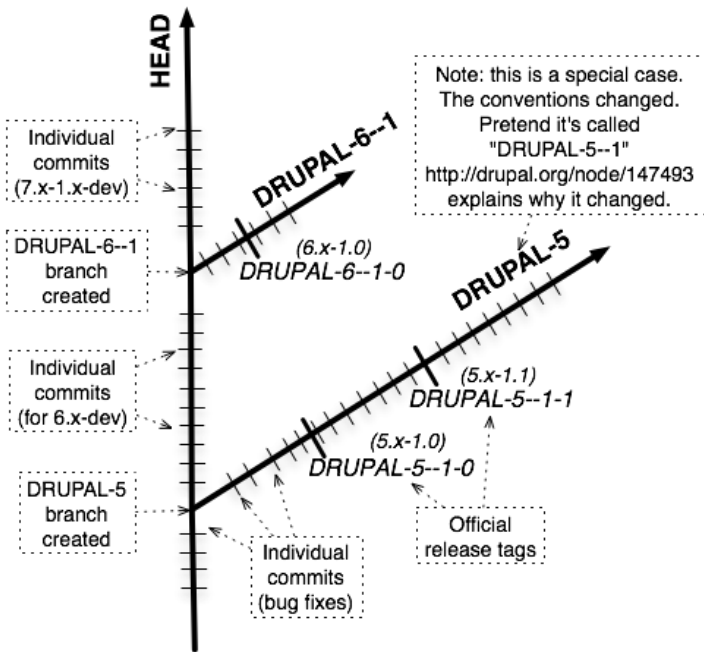
- Revision: a specific copy of something. For example, a particular file in a given state. Every time you commit a change, you get a new revision.
- Branch: an isolated set of revisions that are independent of other branches. Imagine this is a new directory with its own copy of each file. You can modify files in one directory without changing the copies in others.
- Tag: a label or name you give to a given set of revisions of the files on a certain branch.

CVS Branches and Tags for Drupal Core



Why using branches matters

- Branches allow you to isolate changes and develop for different versions of core.
- Having stable (bug-fix-only) branches let you keep part of your code stable so people using it can run their sites, while you work on cool new features without fear or hesitation.
- Development snapshot release nodes for your branches allow testers without CVS to access your code.
- Release notes can explain your intentions.



How to commit a patch

Before you commit a patch, you should:

- 1) Ask yourself what branch(es) this patch should go in (new feature vs. bug fix)
- 2) Make sure you're working on the branch you think you are [`cvstatus`].
- 3) Make sure your copy only has the changes you think it does [`cvsdiff`].
- 4) Decide who deserves credit.
- 5) Write a short but clear commit message that refers to the issue number, gives credit, and summarizes/justifies the change. (See <http://drupal.org/node/52287> for more).

Why official releases are better

- An official release is based on a specific tag which never changes, so everyone knows exactly what the code is.
- If/when someone reports a bug, you can use the same code to reproduce and fix it.
- Release notes summarize the user-visible changes without the full commit history.

How and when to make a release

- Make a release when you think it's worth it for your users to upgrade.
- Do *not* make a new release after every commit to your code -- don't cry wolf.
- Security fixes (after you've worked with the security team) get an immediate release. <http://drupal.org/security-team> for more.
- Have to use some judgement and sense.
- Beta/RC releases can be very useful.

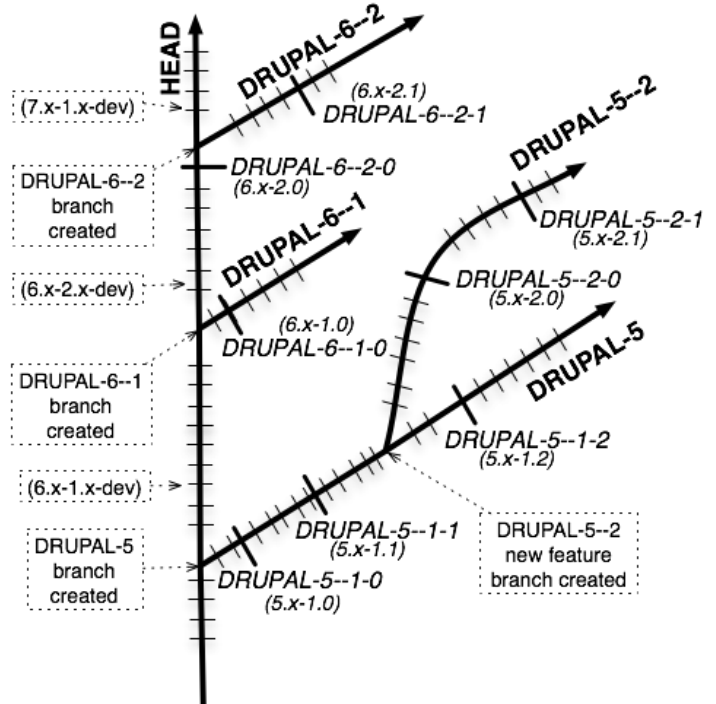
Releases and Update status notifications

// todo

Strategies for using CVS HEAD

There are two primary approaches:

- 1) Keep changing HEAD to follow changes to the in-development version of Drupal core. This helps flesh out potential problems with changes to the core API, and you're ported as soon as the new core is out.
- 2) Use HEAD to write new features for an older, stable version of core. For example, a 6.x-2.* release series while your stable 6.x code is in the DRUPAL-6--1 branch



Other resources

- <http://drupal.org/handbook/cvs>
- <http://drupal.org/handbook/cvs/releases>
- <http://drupal.org/handbook/cvs/quickstart>
- <http://drupal.org/patch>
- CVS: [/contributions/tricks/cvs-release-notes](http://drupal.org/contributions/tricks/cvs-release-notes)