What I've learned about DVCS

(and why I'm still on the fence)

Back in July I wrote a blog post about DVCS

http://tinyurl.com/whydvcs

The two things about DVCS

- Better merging
- Local/offline commit

If SVN released with...

- Better merging
- Local change tracking

What would be left in favor of DVCS?

I started using DVCS anyway

The good news

- Mercurial's really easy to use
- Bitbucket's better than Google Code Hosting

The bad news

- I still don't know whether DVCS is "better"
- I do know some ways in which it's "worse"

Django 1.1

- There was a feature scheduled for this release
- I lost track of it, and now we're behind by over a month

Where is the code?

- The code is in the SVN repository...
- And on GitHub...
- And on Bitbucket...
- And on Launchpad...

Amongst our repositories are...

http://tinyurl.com/darkdvcs

Personal DVCS success

- I use Mercurial at my day job, and in working with Django
- All the repositories I ultimately commit to are SVN

Django DVCS success

- We have committers who use Git
- We have committers who use Mercurial
- We have committers who use Bazaar

All push to the central Django SVN repo

So what, exactly, has been “distributed”?

How do big projects do it?

- Perl before DVCS: "Perl" is whatever Larry says it is
- Perl after DVCS: "Perl" is whatever Larry says it is

Successful DVCS use doesn't seem to use the "D" much

So we come full circle

- Better merging
- Local/offline commit
- If SVN sprouted these, what would be left in favor of DVCS?
- Easy repository creation?

Where I am today

- I like better merging, and *anything* is better than SVN's merging
- But that doesn't require DVCS, and DVCS has some issues to work out

Maybe we just haven't found the right patterns

So SVN isn't quite dead yet (but it's not getting better)