A Brief Introduction to Distributed Version Control

Kyle Cordes http://kylecordes.com
SLUUG October 10, 2007

You Branch and Merge.

A fundamental truth: every time you edit a file you are branching, and every time you reconcile with another developer you are merging. In most tools you get one easy branch and merge locus: your local working directory. All other branching is a Big Deal.

It does not have to be this way.

A Short Tour of 3 tools

bzr http://bazaar-vcs.org/

hg http://www.selenic.com/mercurial/

git http://git.or.cz/

Also, check your distro's package system.

What is a DVCS? Why?

- Peer-to-peer design
- Everyone gets all the features, rather than the interesting features limited to a high priest class.
- Make use of the massive CPU and disk capacity on dev machines.
- No central server needed, though many projects nominate a machine for this purpose.
- Use a "dumb" storage location for a repository, if desired. Or a "smart server" for performance and security.
- Work offline, with full history, branches, merges.
- No central administrator is needed, potentially a cost savings.
- Very cheap branching, in some cases immediate, even for large projects.
- Very good merging, because you merge all the time.
- Commit-then-merge, not merge-then-commit.
- Repeated merge without havoc.
- Merge keeps both sides of history, which is important because you merge a lot. This varies by tools, for example apparently bzr keeps this history less effectively than some others.
- Depending on what you are coming from and which tool you choose, the speed gain can be so remarkable that it help every developer every day.

Some SVN Nitpicks

It is easy and tempting to pick on SVN. Linus does so vigorously in his online talk. I don't hate it as much as he does but, these things bother me:

- SVN is slow for large projects.
- Branching in SVN sounds clever as you read its cheap copy story. It's a great story. But actually using it is ridiculous; both in the URL-crazy syntax and utter lack of merge history.
- We don't need a better CVS, we need something much better than CVS.
- .svn directories scattered all over are a pointless pain.
- .svn directories are enormous, sometimes larger than the entire project history in git or hg!

Security

Security is a weak area in terms of out-of-the-box features and tutorials, because these tools come from the open source world where the default is for everyone to be able to see all code. However, with a little effort you can set up whatever security you like:

If you're serving over HTTP, you can use Apache mod_whatever to control access.

Tunnel over SSH (in the box, in most cases) to avoid ever sending code in the clear.

Even a "dumb" storage location can be secured with SFTP.

Scripts can be used for per-branch and other fine grained access control, akin to what you can do with svn-access.conf. There are examples online.

I Can't Use a DVCS Because:

SOX/HIPPA/CMM/etc. requires a centralized tool.

SOX/HIPPA/CMM/etc, is the standard reason why anyone can't do anything. Some of these tools facilitate much stronger guarantees about the provenance of the source code than you get from a centralized tool, because they have credible and straightforward ways to verify that centralized store has not been compromized.

We are all in one place, therefore a DVCS makes no sense.

Actually many of the features in these tools are as useful in the same building as in a worldwide team.

Tool ZZZ is our corporate standard.

Then you should use it, don't get fired. However, many people are using a DVCS in front of their corporate standard tool.

DVCS vs DVCS:

Many DVCS tools treat each repo/workspace as a branch and vice versa, so if you use many branches you will have many workspaces.

bzr can use shared files to reduce the bloat from this.

git handles many branches much better, with arbitrarily many per repo/workspace.

My feeling is that hg and git are more mature than bzr.

git does not work well on windows yet.

Other DVCS to Consider

Monotone – has some slick features, but does not appear to scale well to large projects. It stores information in a SQLite DB. Monotone, unlike any others I've seen, replicates all branches by default, which is nice.

> An aside – there are fascinating thoughts on how to a data synchronization system can work, in the Monotone docs / presentations.

arch / tla – one of the early DVCSs that started all this. I have heard it is mystifying to use.

darcs – everyone loves its "cherry picking", but I have not triedit.

SVK – optimized for being a better svn client, with offline history and merging that works. Stores merge history in SVN attribute.

More git Merging

Depending on time, I will show more of the branch / merge facilities in git, as well as its gitk GUI.

Other Resources

http://wiki.sourcemage.org/Git_Guide
http://www.adeal.eu/

Task	git	bzr	hg
Big Name	Linus	Mark Shuttleworth	Matt Mackall (Who'se he?)
Identify yourself	git config user.name "FirstName LastName" git config user.email "user@example.com"	GUI tool, or ~/bazaar.conf	~/mercurial.ini
Start a new project	git init	bzr init	hg init
See what you might add	git add -n -v .	bzr add --dry-run	hg add -n
Configure what to skip	.gitignore	.bzrignore	.hgignore
Add original set of files to source control	git add .	bzr add	hg add
Commit additions / changes	git commit -m "message"	bzr commit -m "message"	hg commit -m "message"
See log	git log	bzr log	hg log
Add new files	git add .	bzr add	hg add
Commit changes to changed files	git commit -a -m "message"	bzr commit -m "message"	hg commit -m "message"
get an existing project	git clone URL	bzr clone URL	hg clone URL
push your changes upstream	git push	bzr push URL	hg push
get up to date	git pull	bzr pull	hg pull