Keeping up with Koha

Robin Sheat <robin@catalyst.net.nz>

31st October 2011

Where This Talk Began

What Came Up?

- People are stuck on version 2.2!
  - (that came out January 6, 2005 - over 6 years ago)
- They wanted to know how to upgrade, and how to stop this happening again.

Well, What's the Answer?

Upstreaming

What We're Going to Cover

- Introduction
- About Upstreaming
- Problems With Not Upstreaming
- Benefits of Upstreaming
- When Not to Upstream
- How To Upgrade
- The Process of Upstreaming
- How to Avoid Slipping Behind

What Is Upstreaming?

What Is Upstreaming?

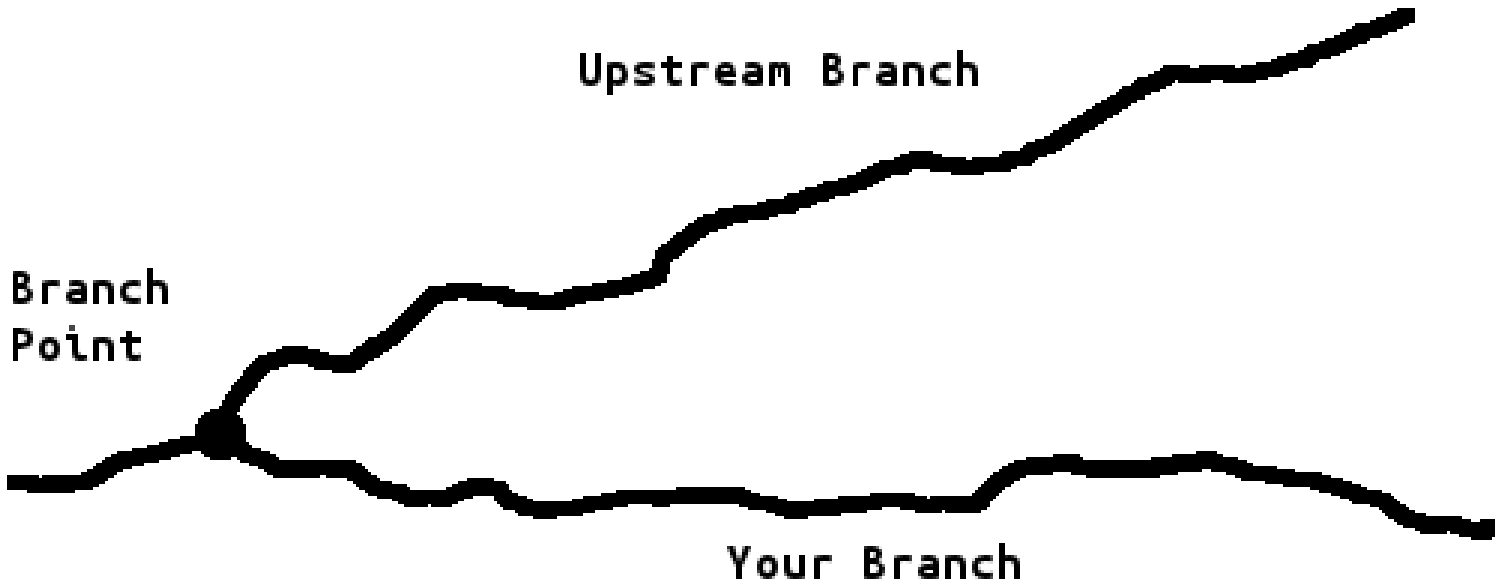
- Upstreaming is the process of taking all your local customisations and sending them back to the project.
- It's pretty common in the Free Software world.

If it's such a great idea...

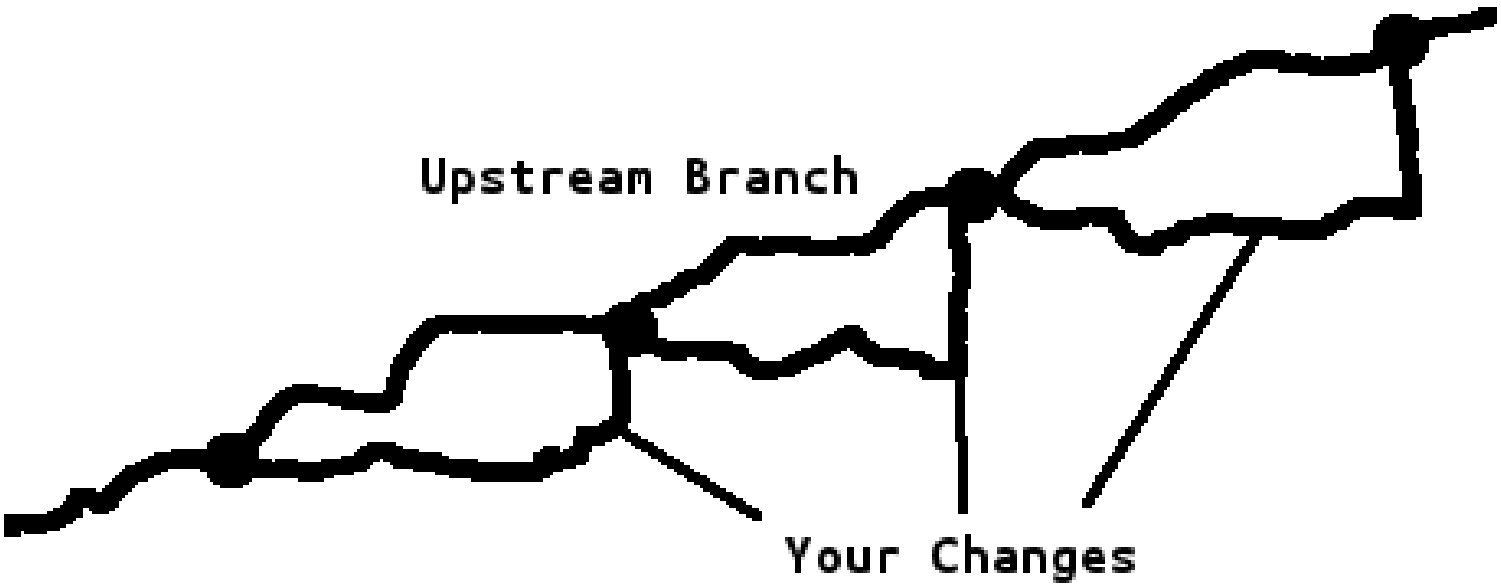
...why doesn't everyone do it?

- People don't know how
  - ...come to the hackfest!
- It takes time
  - ...it's true, it does.
- People (usually management) don't understand Free Software licensing
  - ...this doesn't seem to apply to libraries.

What Not Upstreaming Looks Like



What Upstreaming Looks Like



Problems With Not Upstreaming

It's hard to upgrade

- Miss out on all the cool new features.
- Lots of work when you find you do need to upgrade.
- Have to manually keep an eye on anything important, e.g. security patches

Problems With Not Upstreaming

The further away you get from being current, the harder it is to upgrade.

The biggest cause of this is customisations.

Aside: About Koha Release Cycles

- Small releases every month
- Big releases every six months
- A lot can change in a very short space of time, making it harder to keep up.

Benefits of Upstreaming

Makes Upgrading Easier

- Keeping up to date will be harder if you have your own changes.
- Upgrading with customisations causes:
  - Conflicts
  - A need to re-test
  - New features replacing your customisations

Benefits of Upstreaming

Everyone Gets to Use Your Features

- Libraries don't have competitive advantage with their ILS
- Credit!
- They're supported by other people
  - No more maintenance!

Benefits of Upstreaming

Become Part of the Community

- Get to try/test features before they're released
- Get to meet cool people at conferences

When Not to Upstream

It's Not Always the Way

Sometimes you shouldn't upstream.

When Not to Upstream

Really Specific Customisations

For example:

- integrating with an intranet
- custom authentication systems
- things that might break the specifications

When Not to Upstream

Really Small Changes

- Upstreaming may be more trouble than it's worth
- Unless others might find it useful
  - More common than you might think
  - E.g. "fines" changing to "fines and charges"

When Not to Upstream

Sometimes It May Be Worth Upstreaming Anyway...

- Features can often be generalised to work for other people
- Make a syspref or two that allow it to be configured

When Not to Upstream

In case you really can't upstream something...

- Do it in Git
- This will allow you to merge upstream changes in easier.
  - Learn to rebase: `git rebase 3.6.1`
- Update regularly
  - Smaller upstream changes more often are easier to deal with than big ones every so often
- Watch for big upcoming changes
  - Like the switch to Template::Toolkit
- The change between releases (e.g. 3.6 to 3.8) will always be hard

How To Upgrade

So you've decided to upgrade...
...but you're out of date, and have customisations.

There are really two options:

How To Upgrade

Port your changes

- If they're big, this can be a lot of work.
  - Perhaps there's something similar in the newer versions.
- If they're small, it might not be hard.
- Support companies can help with this.

How To Upgrade

Throw it all away, start again

- But keep the data!
  - This is quite easy if you're on a version 3 series already.
  - Harder from 2 and lower.
  - Too many people are stuck on 2.2.
- Likely to be the best method for large changes
  - Now you can reimplement the changes you need, and upstream them!

The Process of Upstreaming

Different Approaches

- Do everything in-house, throw it over the wall when finished.
  - Often the approach taken by companies who don't "get" Free Software.

The Process of Upstreaming

Different Approaches

- Work with the community every step of the way, uploading every change.
  - This is the ideal.
  - Not always practical.

The Process of Upstreaming

Different Approaches

- Best way: a mix of these approaches
- For little things, just do it.
- For huge things, ask for comments, help, write an RFC, etc.
- For things in the middle, find people who will help and give you feedback.

The Process of Upstreaming

Regarding Git

Use Git

No really. Always use Git.

The Process of Upstreaming

Prepare the Patch

- Create a ticket at the Koha bugtracker.
- Follow the coding guidelines.

The Process of Upstreaming

Prepare the Patch

- Prepare it against master.

"Do NOT fall into the trap of adding more and more stuff to an out-of-tree project. It just makes it harder and harder to get it merged. There are many examples of this."
— Andrew Morton (Linux Kernel Developer)

The Process of Upstreaming

Prepare the Patch

- If it's a bug: also make a patch against the current stable version.
- If it changes behaviour: hide it behind a system preference.
- Make sure it updates the database schema, and updatedatabase.pl.

The Process of Upstreaming

Submit the Patch

- Attach your patch to the bug, send it to the mailing list.
- Revise it in response to comments.
- Be prepared to throw it away if something similar or better comes along.
- For medium-sized changes, be prepared for a 20% or 30% overhead when upstreaming.
  - Unless it takes ages to get in, then it might need a lot of refactoring.

How to Avoid Slipping Behind

If you're implementing a new feature...

- Release early, release often.
- Work in self-contained units (where possible.)
- Get feedback from other developers.
- Try to find others to help.
- Try to get other libraries involved.
  - Pooling resources is a big strength of Free Software.

How to Avoid Slipping Behind

Try to keep up to date

- Always develop off master
  - Perhaps run a testing system with master so you can see what's happening there.
- Maintain your features
  - You'll have to keep them up to date until they get into a stable branch.
- Keep your production versions fairly up to date.

In Conclusion

- Get up to date.
- Stay there.
- Work with the community.
- Accept that it'll seem like more work, but probably won't really be.