

Best Practices in Maintaining Vendor-Specific GTK+ branches

Kristian Rietveld

<kris@gtk.org>

<kris@lanedo.com>

Overview

- Why?
- How not to do it
- Ideal-world vendor branches
- Case study: Maemo-GTK+
- Conclusions

Why?

- Why do people do this?
 - To provide additional features
 - To change things that they didn't like
 - Adhere to UI specifications

How do people do this?

1. Download upstream tarball.

How do people do this?

1. Download upstream tarball.
2. Unzip.

How do people do this?

1. Download upstream tarball.
2. Unzip.
3. Make changes.

How do people do this?

1. Download upstream tarball.
2. Unzip.
3. Make changes.
4. Zip.

How do people do this?

1. Download upstream tarball.
2. Unzip.
3. Make changes.
4. Zip.
5. Done.

Problems

- No easy upstream upgrades
- No bug fixes
- No new features
- Horrible to maintain

Ideal world

- Vendor branch to do the following:
 - Staging area for new features
 - Maintain features that cannot be included upstream (yet)
 - Miscellaneous incompatible or rejected patches

Activities

- Activities around vendor branches:
 - Push all bug fixes upstream
 - Look into upstreaming new features
 - Update upstream version regularly
- **Goal:** Get as much code upstream as possible

Benefits of the goal

- Smaller diff with upstream enables easy upgrades (even without git)
- Upstream maintains features
- For features including API: more polished and complete

Types of changes

- Just adding new widgets
- Modifying existing widgets (if you must):
 - Adding property
 - Change property default
 - Changing drawing code
 - etc.

Modifying existing code

- Always has to happen with care
- Do not remove/change code that you do not fully understand
- Not being careful introduces more bugs

Modifying existing code (cont.)

- Guard your changes in `#ifdef`
- Better overview of what's going on when hacking
- Helpful when merging

Document your changes

- Maintain a Wiki page with descriptions of all change sets
- Is very helpful when:
 - Working on other people's code
 - Identify what could go upstream
 - Merging upstream upgrades

Building on top of upstream

- Consider to make changes in a subclass, possibly in a separate library
- Further reduces diff size
- If impossible for certain cases: file bugs

Version control

- Good version control is a must
- *Worst:* tarball + 1 large patch
- *Past:* not much choice: CVS or Subversion
- *Now:* you want to use git
- *Small scale:* tarball + quilt

ABI implications

- Think twice before breaking ABI
- Complicates upgrades
- Could complicate application porting

Namespacing

- When you add new API to your modified GTK+, avoid the “gtk_” namespace
- Prefix it, for example “maemo_gtk_”
- Unless you are backporting upstream patches

Case study: Maemo-GTK+

- Starting point:
 - Diff against GTK+ 2.6.4
 - No guards
 - No documentation
 - Size: approx 1MB, 30K lines
- **Goal:** upstreaming, make diff smaller

Grasping the diff

- Split diff into diffs per file
- Per file, inventory all change **sets**
- A change set is all changes related to a single feature
- This results in a set of Wiki pages

Listing change sets

- Document which files are touched by each change set
- Often across SVN revisions/git commits
- Add a helpful description **and** use case
- All of this will help when submitting a patch upstream

Start version control

- If you are not using it yet
- Commit all change sets separately
- Automatically starts tracking history

Determine quality & fate

- Change set ok?
 - File bug upstream, attach patch
- Change set not ok?
 - Rewrite and file upstream
 - Keep it internal, maybe refactor
 - Remove it completely
 - Move to new library
- Maintain status of change set in Wiki

Progress

- By the time we upgraded to GTK+ 2.10, the diff lost a third of its size
- Version upgrades are vital in this process
- Continue until each change set has been processed
- Apply this scheme to new developments as well!

When done

- By the time we were done:
 - Large number of features upstreamed (e.g. theming, keyboard navigation)
 - Size of diff decreased
 - Could do 2.16->2.18 upgrade with 1 person in 1 day
 - Most new patches went into Maemo-GTK and upstream simultaneously

Conclusions

- Vendor branches do serve a purpose
- Structured approach is required to make it a success
- Maintenance time can be substantially reduced
- Upstream benefits from your bug fixes & features

Questions?