Source control/management of implementation of an existing open source project [closed]

Asked 16 years, 1 month ago Modified 16 years, 1 month ago

Viewed 185 times

3

Closed. This question needs to be more focused. It is not currently accepting answers.

Want to improve this question? Update the question so it focuses on one problem only by editing this post.

Closed 9 years ago.

Improve this question

I'm currently installing and configuring an open source project on a test server. I've been well trained to use source control, and I want to make sure everything I do is managed properly (both for me, the sole dev/admin, and for future maintaners). The open source project is available as a source download or by svn checkout.

I want to have my own source controlled version of the project. I don't intend to be changing the (java servlet) code much (if at all), but there are configurations, XML

files, XSL, CSS, etc. all involved that I definitely want to be source controlled.

Should I go ahead and just make my own local repository of all of the source code? Should I try to only control the files that I need to change? In that case I would want the directory structure to match, so I could do checkouts directly to the build directories.

version-control open-source

Share

Improve this question

Follow

edited Oct 30, 2008 at 13:50

Till

22.4k ● 4 ● 60 ● 89

asked Oct 30, 2008 at 13:34

Jonathan Adelson

3,321 ● 5 ● 32 ● 39

3 Answers

Sorted by: Highest score (default)

1

Having two checkouts to the same directory tree won't work. If you check out your source, and try to check out the OSS project source, any directory they have in common will fail, saying it's already a working directory for a different project.

If you can gather the css, xml, xsl etc. into a common directory, you can put those in a single directory in your

own project's svn, and then check them out into a directory in the working directory of the OSS project.

~/Working => svn://samhoice/project/trunk

~/Working/osscomponent => svn://osshost/project/latesttag

~/working/osscomponent/config => svn://samhoice/project/trunk/config

In this structure, the osscomponent directory does not exist in samhoice's svn repository. It is added by your setup script as the working directory root of the OSS project. The config directory is not checked out from the OSS project, and doesn't exist there. The config directory is created by your setup script and the config directory is checked out there from your project repository.

So in this directory structure you have three checkouts. There is no recursive overlap, so you have no conflicts between svn mappings on any subdirectories.

If you need your config files to be arranged within the structure of the OSS project, add some symlinks with your makefile or config script. You can also probably do this in a post-checkout hook in your svn client.

I use a structure like this for one of my projects, for sharing some code between two project trees. The shared stuff is in a subtree like I recommended for your config section.

Share Improve this answer

Follow

answered Oct 30, 2008 at 13:53

Mnebuerquo

5,939 ● 6 ● 48 ● 54

1

I would check in all code, because even though you don't want to change them or don't see a reason to right now, someone else might see a reason later on. Also, there might be a bug, or simple refactoring to improve maintainability or performance, etc..

A basic layout for SVN would be:

```
branches/
tags/
trunk/
```

And the general directory layout (underneath those three) should of course match your application so you can check it out and run it right away. You could however do slight modifications through a build file, etc.. But I guess *checkout and go* is always nice. For more information, or pointers on best-practices I would look at other opensource projects using your technology.

As for configuration files - I wouldn't check those in. Most projects however check in a `foo-dist` file which shows the user all configuration options. At the minimum, they would need to copy `foo-dist` to `foo` to enable the default configuration.

Also, have you looked at Google Code, Sourceforge or github for project hosting including version control? All three offer you a ton, for free - also free in terms that you don't need to take care of a server for your project hosting, etc..

Share Improve this answer

Follow

answered Oct 30, 2008 at 13:49

Till

22.4k ● 4 ● 60 ● 89

The question asker specifically wants his config files in svn. – Mnebuerquo Oct 30, 2008 at 14:03

I do want configurations in svn, but some of them can't be stored there because they include site specific data that can't be available. – Jonathan Adelson Oct 30, 2008 at 17:11

1

I've used several methods, depending on the size of, and amount of customization to, the 3rd-party project:

- **For small or highly-customized projects** I'll check the whole thing in, starting with the stock code. This makes it easy to work on the entire project and provides "one-stop shopping" to check out everything I need to build it.
- **For large or slightly-customized projects** I'll keep a copy of the stock archive (typically a `.tar.bz2` or `.zip` file) and be sure it's backed up. Then I'll keep the files that embody my customizations under revision control in some form. Options here are

    - Check in (only) the customized files themselves. This is usually the best choice if you're modifying source code.
    - Check in a script that does the customization. This is most effective if the script supplies input to a configuration utility, because you're storing your customizations (the input data) separate from the stock pieces (the config utility). If a new version of the 3rd-party library is released, it's usually easy to apply your customizations to it, allowing for critical upgrades.
- **For projects that "will never change™,"** or for which customers have **contractually bound you to particular versions,** store the stock code somewhere, and create patch files under revision control. Then you can release the "approved" version with a patch. *Note: this is a fairly horrible choice, as patch files are awful to maintain--if you make a new patch, you must either re-create the patch files or create an additional patch that's applied on top of the first patch. The reasons for choosing this option are usually not chosen for technical reasons.*

I cannot stress enough that _**you need to document the entire process of building this sort of project and adding customizations, or automate it, or (preferably) both.** It's astonishingly easy to forget which files belong where, or to whom, and it's painful to start over. Especially if you're the heir.

Good luck!

Share Improve this answer

Follow

answered Oct 30, 2008 at 22:56

Adam Liss
48.3k 13 113 152