The theory (and terminology) behind Source Control

Asked 16 years, 4 months ago Modified 12 years, 4 months ago

Viewed 1k times

7

I've tried using source control for a couple projects but still don't really understand it. For these projects, we've used TortoiseSVN and have only had one line of revisions. (No trunk, branch, or any of that.) If there is a recommended way to set up source control systems, what are they? What are the reasons and benifits for setting it up that way? What is the underlying differences between the workings of a centralized and distributed source control system?

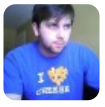
version-control

Share

Improve this question

Follow

edited Aug 17, 2008 at 2:05



Ryan Fox

10.2k ● 5 ● 39 ● 48

asked Aug 17, 2008 at 1:38

Cristián Romo

10k ● 12 ● 52 ● 50

1 Such a nice question! Why is not it closed yet by the moderators running hungry for anything more interesting than

the 1B "How do I X in Java?" questions? :) – mlvljr Feb 15, 2015 at 6:47

5 Answers

Sorted by: Highest score (default)

8

Think of source control as a giant "Undo" button for your source code. Every time you check in, you're adding a point to which you can roll back. Even if you don't use branching/merging, this feature alone can be very valuable.

Additionally, by having one 'authoritative' version of the source control, it becomes much easier to back up.

Centralized vs. distributed... the difference is really that in distributed, there isn't necessarily one 'authoritative' version of the source control, although in practice people usually still do have the master tree.

The big advantage to distributed source control is two-fold:

1. When you use distributed source control, you have the whole source tree on your local machine. You can commit, create branches, and work pretty much as though you were all alone, and then when you're ready to push up your changes, you can promote them from your machine to the master copy. If you're working "offline" a lot, this can be a huge benefit.
2. You don't have to ask anybody's permission to become a distributor of the source control. If person

A is running the project, but person B and C want to make changes, and share those changes with each other, it becomes much easier with distributed source control.

Share Improve this answer Follow

edited Aug 17, 2008 at 1:43

answered Aug 17, 2008 at 1:39

Brad Wilson

70.4k ● 9 ● 77 ● 85

6

I recommend checking out the following from Eric Sink:

http://www.ericsink.com/scm/source_control.html

Having some sort of revision control system in place is probably the most important tool a programmer has for reviewing code changes and understanding who did what to whom. Even for single person projects, it is invaluable to be able to diff current code against previous known working version to understand what might have gone wrong due to a change.

Share Improve this answer Follow

answered Aug 17, 2008 at 1:44

Tall Jeff

9,984 ● 7 ● 46 ● 61

Here are two articles that are very helpful for understanding the basics. Beyond being informative,

3

Sink's company sells a great source control product called Vault that is free for single users (I am not affiliated in any way with that company).

http://www.ericsink.com/scm/source_control.html

http://betterexplained.com/articles/a-visual-guide-to-version-control/

Vault info at www.vault.com.

Share Improve this answer

Follow

edited Aug 11, 2012 at 15:42

burning_LEGION

13.4k ● 8 ● 41 ● 52

answered Aug 17, 2008 at 1:47

roger pence

1

Even if you don't branch, you may find it useful to use tags to mark releases.

Imagine that you rolled out a new version of your software yesterday and have started making major changes for the next version. A user calls you to report a serious bug in yesterday's release. You can't just fix it and copy over the changes from your development trunk because the changes you've just made the whole thing unstable.

If you had tagged the release, you could check out a working copy of it and use it to fix the bug.

Then, you might choose to create a branch at the tag and check the bug fix into it. That way, you can fix more bugs on that release while you continue to upgrade the trunk. You can also merge those fixes into the trunk so that they'll be present in the next release.

Share Improve this answer

Follow

answered Aug 17, 2008 at 1:47

Matt Miller

3,499 ● 5 ● 29 ● 26

1

The common standard for setting up Subversion is to have three folders under the root of your repository: trunk, branches and tags. The trunk folder holds your current "main" line of development. For many shops and situations, this is all they ever use... just a single working repository of code.

The tags folder takes it one step further and allows you to "checkpoint" your code at certain points in time. For example, when you release a new build or sometimes even when you simply *make* a new build, you "tag" a copy into this folder. This just allows you to know exactly what your code looked like at that point in time.

The branches folder holds different kinds of branches that you might need in special situations. Sometimes a branch is a place to work on experimental feature or features that might take a long time to get stable (therefore you don't want to introduce them into your main line just yet). Other times, a branch might represent the "production" copy of your code which can be edited and deployed

independently from your main line of code which contains changes intended for a future release.

Anyway, this is just one aspect of how to set up your system, but I think giving some thought to this structure is important.

Share Improve this answer

Follow

answered Aug 17, 2008 at 1:51

jeremcc

8,733 ● 12 ● 46 ● 55