# Why is branching and merging easier in Mercurial than in Subversion? [closed]

Asked 16 years, 3 months ago Modified 14 years, 5 months ago

Viewed 12k times

92

**Closed**. This question is opinion-based. It is not currently accepting answers.

**Want to improve this question?** Update the question so it can be answered with facts and citations by editing this post.

Closed 7 days ago.

Improve this question

Handling multiple merges onto branches in Subversion or CVS is just one of those things that has to be experienced. It is inordinately easier to keep track of branches and merges in Mercurial (and probably any other distributed system) but I don't know why. Does anyone else know?

My question stems from the fact that with Mercurial you can adopt a working practice similar to that of Subversions/CVSs central repository and everything will work just fine. You can do multiple merges on the same branch and you won't need endless scraps of paper with commit numbers and tag names.

I know the latest version of Subversion has the ability to track merges to branches so you don't get quite the same degree of hassle but it was a huge and major development on their side and it still doesn't do everything the development team would like it to do.

There must be a fundamental difference in the way it all works.

svn git version-control mercurial

Share

Improve this question

Follow

edited Oct 3, 2008 at 16:12

asked Sep 4, 2008 at 15:16

Nick Pierpoint

17.8k ● 9 ● 47 ● 74

## 6 Answers

Sorted by: Highest score (default)

115

In Subversion (and CVS), the repository is first and foremost. In git and mercurial there is not

> really the concept of a repository in the same way; here changes are the central theme.

+1

The hassle in CVS/SVN comes from the fact that these systems do **not** remember the parenthood of changes. In Git and Mercurial, not only can a commit have multiple children, it can also have multiple parents!

That can easily observed using one of the graphical tools, `gitk` or `hg view`. In the following example, branch #2 was forked from #1 at commit A, and has since been merged once (at M, merged with commit B):

```
o---A---o---B---o---C         (branch #1)
     \       \
      o---o---M---X---?       (branch #2)
```

Note how A and B have two children, whereas M has two **parents**. These relationships are *recorded* in the repository. Let's say the maintainer of branch #2 now wants to merge the latest changes from branch #1, they can issue a command such as:

```
$ git merge branch-1
```

and the tool will automatically know that the *base* is B--because it was recorded in commit M, an ancestor of the tip of #2--and that it has to merge whatever happened between B and C. CVS does not record this information, nor did SVN prior to version 1.5. In these systems, the graph would look like:

```
o---A---o---B---o---C         (branch #1)
     \
      o---o---M---X---?       (branch #2)
```

where M is just a gigantic "squashed" commit of everything that happened between A and B, applied on top of M. Note that after the deed is done, there is *no trace left* (except potentially in human-readable comments) of where M did originate from, nor of *how many* commits were collapsed together--making history much more impenetrable.

Worse still, performing a second merge becomes a nightmare: one has to figure out what the merge base was at the time of the first merge (and one *has* to *know* that there has been a merge in the first place!), then present that information to the tool so that it does not try to replay A..B on top of M. All of this is difficult enough when working in close collaboration, but is simply impossible in a distributed environment.

A (related) problem is that there is no way to answer the question: "does X contain B?" where B is a potentially important bug fix. So, why not just record that information in the commit, since it is *known* at merge time!

P.-S. -- I have no experience with SVN 1.5+ merge recording abilities, but the workflow seems to be much more contrived than in the distributed systems. If that is indeed the case, it's probably because--as mentioned in the above comment--the focus is put on repository organization rather than on the changes themselves.

Share Improve this answer Follow

answered Sep 4, 2008 at 20:32
Damien Diederen
2,474 ● 1 ● 17 ● 7

6 SVN 1.5+ would indeed put an SVN property on the merge commit M listing the merged commits it contains, in other words A-B. So you have the same information. – Simon D Mar 11, 2010 at 11:27

Kind of. Technically, SVN's merge tracking is akin to what Git calls "cherry-picking," with additional magic to make it easier on the user; it is semantically different from what Git, Hg and Bzr do when merging. I guess it does not make a lot of difference in practice as long as you do not care about the DAG (eagain.net/articles/git-for-computer-scientists). – Damien Diederen Mar 22, 2010 at 10:35

I assume the SVN 1.5 feature you're talking about is the use of the `svn:mergeinfo` property? – Tyler Mar 10, 2011 at 7:34

@MatrixFrog: Yes, this is just what `svn:mergeinfo` does. – sleske Oct 28, 2011 at 9:34

@CharlieFlowers You can always award a bounty! – Josiah Yoder Feb 26, 2021 at 20:20

Because Subversion (at least version 1.4 and below) doesn't keep track of what have been merged. For Subversion, merging is basically the same as any commit

12

while on other version control like Git, what have been merged are remembered.

Share Improve this answer Follow

answered Sep 4, 2008 at 16:02

htanata
36.9k ● 8 ● 51 ● 57

---

7

Untouched by any of the already provided answers, Hg offered superior merge capabilities because it uses more information when merging changes (hginit.com):

> For example, if I change a function a little bit, and then move it somewhere else, Subversion doesn’t really remember those steps, so when it comes time to merge, it might think that a new function just showed up out of the blue. Whereas Mercurial will remember those things separately: function changed, function moved, which means that if you also changed that function a little bit, it is much more likely that Mercurial will successfully merge our changes.

Of course, remembering what was last merged (the point addressed by most of the answers provided here) is also a huge win.

Both improvements, however, are questionable since subversion 1.5+ stores additional merge information in the form of subversion properties: that information available, there's no obvious reason why subversion merge couldn't implement merge as successfully as Hg or Git. I don't know if it does, though, but it certainly sounds like subversion developers are on their way to get around this issue.

Share Improve this answer

Follow

answered Jul 5, 2010 at 13:27

Tomislav Nakic-Alfirevic

**10.2k** ● 5 ● 40 ● 51

---

4

I suppose this might partially be because Subversion has the idea of a central server along with an absolute time line of revisions. Mercurial is truly distributed and has no such reference to an absolute time line. This does allow Mercurial projects to form more complicated hierarchies of branches for adding features and testing cycles by sub project however teams now need to much more actively keep on top of merges to stay current as they can't just hit update and be done with it.

Share Improve this answer

Follow

answered Sep 4, 2008 at 15:55

Evil Andy

**1,748** ● 1 ● 14 ● 19

---

3

In Subversion (and CVS), the repository is first and foremost. In git and mercurial there is not really the concept of a repository in the same way; here *changes* are the central theme.

I've not thought much about how you'd implement either but my impression (based on bitter experience and lots of reading) is that this difference is what makes merging and branching so much easier in non-repository based systems.

Share Improve this answer

Follow

answered Sep 4, 2008 at 15:56

Stephen Darlington
52.5k ● 12 ● 108 ● 153

1

I only have experience with Subversion but I can tell you that the merge screen in TortoiseSVN is horribly complicated. Luckily they include a dry run button so that you can see if you are doing it right. The complication is in the configuration of what you want to merge to where. Once you get that set up for the merge the merge generally goes fine. Then you need to resolve any and all conflicts and then commit your merged in working copy to the repository.

If Mercurial can make the configuration of the merge easier then I can say that would make merging 100% easier then Subversion.

Share Improve this answer

Follow

answered Sep 4, 2008 at 15:40

RedWolves
10.4k ● 12 ● 50 ● 69