TOOLS

Three-Way Merging: A Look Under the Hood

By Pablo Santos, December 24, 2013

Post a Comment

Automating three-way code merges requires considerable sophistication from the version control system.

Three-Way Merge Tool Layout

When you run a three-way merge tool, the typical layout of the tool is as illustrated in Figure 5:

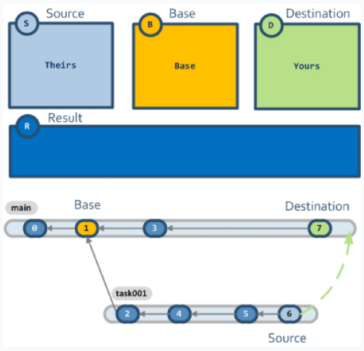

Figure 5.

Good three-way merge tools show four panels:

- "Theirs" (the source of the merge, see the branch diagram in Figure 6), base, and "Yours" (the destination of the merge) in the upper panel.
- The result of the merge in the lower panel.

To me this four-panel representation of the three-way merge is the most intuitive, but some tools present this alternative layout with only three panels (Figure 6):

Figure 6.

In this layout, the "destination/yours" (or working copy) and the result of the merge are displayed together.

The Importance of Merge Tracking

To run effective three-way merges, you not only need a good three-way merge tool, you need an effective merge engine in your version control tool.

In fact, part of the mission of version control should be to correctly calculate the common ancestor/base on any three-way merge. When people say "git is very good at merging," what they mean is "git is very good tracking the merge history, hence calculating the common ancestor for each file." In my VCS work, we put a lot of effort into the merge engine and calculating the nearest common ancestor.

Let's go back to the three-way merge with a manual conflict that we just solved, and let's check out the branching structure (well, at least one very simple branching structure); see Figure 7:

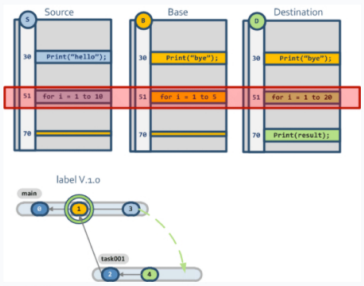

Figure 7.

- Changeset 3: someone working on the "main" branch performed the change of the `Print("hello")` line
- Changeset 4: meanwhile, on branch "task001," you were doing the addition of the `Print(result)` at line 70.
- And you both modified line 51.

Now, you want to merge the latest changes coming from "main" into your branch "task001." The version control system will find the nearest common ancestor of changesets "3" and "4" and it will use the graph above. The result in this simple case is changeset "1." The "base" version will be retrieved from changeset "1" to do the merge.

Once you solve the manual conflict on line 51, you will be checking in on the branch "task001" and creating a new changeset "5" as in Figure 8:

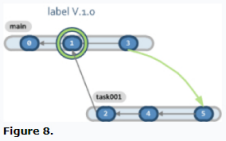

Figure 8.

Now development continues; somebody will be creating more changes on "main" while you perform a new checkin on "task001." And then you decide you have to merge "task001" back to "main" (Figure 9):

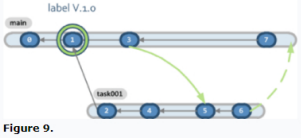

Figure 9.

The version control system will have to calculate the base/common ancestor between "6" and "7."

The common ancestor will be "3" as Figure 10 shows:

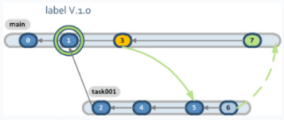

Figure 10.

Note that "3" is the common ancestor because the version control is considering the merge that happened between "3" and "5," which you completed before.

What is the benefit of this merge tracking?

Well, if the merge link between "3" and "5" wasn't tracked (as used to happen with old version control systems), then the base would be "1" again, and you would have to again solve the manual conflict you already solved before. However, if the version control system does its job correctly, the ancestor will be identified as "3" and you won't have to waste time on conflicts you already solved.

Now, what would happen here with two-way merge? You would have to solve *every* difference manually because in two-way merges, every single modification is a conflict since the merge facility doesn't have a way to solve conflicts automatically.

Conclusion

Often, the questions regarding three-way merge are asked by developers using version control systems lacking good merge tracking such as CVS, Microsoft Visual SourceSafe, and even old versions of Subversion.

Understanding how three-way merge works and why it is so important to have a good merge engine like those in new distributed version control systems is key when looking for a replacement to an aging SCM.

Pablo Santos is a blogger for Dr. Dobb's *and an expert on the operations of version control systems.*

Previous 1 **2**