Merge Early, Merge Often

Integrating changes in branched development

Dear KV,

When doing merged development, how often should you merge? It's obvious that if I wait too long, then I spend days in merge hell, where nothing seems to work and where I wind up using the revert command more often than commit; but the whole point of branched development is to be able to protect the main branch of development from unstable changes. Is there a happy middle ground?

Merge Daemon

Dear Merge,

For years people have asked me about the “happy middle ground,” not only in branched development but also in many areas. I don't know if you've noticed this, but the tenor of these questions is rarely happy and even more rarely results in anything that could be considered a middle ground.

There are two related themes in dealing with merge purgatory (if it were hell there would be no chance of escape, but from purgatory you might be able to work your way out). The first theme is related to how the software you're working on is developed. If there are well-defined boundaries between the sections of the code, then you really should not have to enter merge purgatory at all, because if you're working on a component, then no one else should be making changes to it; the only differences will be yours, case closed—and well done there. Of course, that's pretty unlikely in any project larger than a few people.

Projects often have the problem of everyone having their fingers in all of the code. That's not difficult to solve if the project is small: you can simply sit down with the other participants in the project and break up the work along some logical boundaries. In a large project you're likely to have several people looking into and modifying the same piece of code. Perhaps one is fixing bugs while another is developing new features, or there are two versions of the product or project that have different teams working from the same main line. Whatever the case, there are a few things you should do to avoid purgatory—such as having an automated build process so that everyone knows when something broken got merged or checked in. For more on this, see my response to Made to be Broken in “Broken Builds” (forthcoming in *Communications of the ACM*, December 2009).

The second and more relevant point to this discussion is the direction of merges. When you're working off in your own branch, it's kind of like working off in a corner of a building—usually a nice quiet place, away from the hubbub of your coworkers where you can concentrate on your work and get things done. Alas, while you're coding in this temporary Nirvana the rest of the world continues on, with all its attendant suffering. It would be nice if you could remain in Nirvana, but, well, you can't.

You have two choices: you can decide to remain in Nirvana as long as possible and then at the very end stride boldly into the mouth of merge purgatory, or you can allow small bits of suffering to enter your world. The suffering, in this case, is a merge from the main branch, the one in which

everyone is committing changes, and from which all of your suffering ultimately comes. When I'm working on branched development—and at this point that's about 95 percent of the time—I merge early and I merge often, just like when I vote.

Now when I say “merge early” I do not mean “early in the morning.” As a matter of fact, I pretty much never say “early in the morning” because I really don't believe such a time exists, and anyone who tries to tell me there is such a time I consider to be a figment of my imagination. In particular, I recommend against merging anyone else's code before you write or debug some of your own. Few things are more frustrating than to start your working day with someone else's broken code messing up your beautiful branch. Get some of the things you wanted to do done first and then integrate a bit of the main branch's suffering into your own. I prefer to do these kinds of merges after I feel good about having gotten some of my own work done. At least then I'm starting out on a high, and the drudgery of fixing my branch will bring me back to a neutral state instead of sending me down into a code depression. I would say that on a fast-moving project you should be merging from the main branch once a day, and no less than once a week. A lot can happen in a week, and you don't want to spend your weekend fixing your working branch.

When is it best to merge code back into the tree? That depends on what you're merging. If it's bug fixes, then they need to be merged as soon as they're tested because there are likely other people depending on those bug fixes. For features they should be complete, which means tested and ready to use. *Complete* does not mean *bug free*. At some point you have to stop polishing that turd and just flush it into the system.

That covers the process part of when to merge, but there is also a tool component. Some source-code control systems simply do not have the tools to support branched development, while others encourage so much branching that code never comes back to the main line. Avoid both of these types of systems if you want to do branched development.

A good system for branched development must include a decent tool for comparing multiple versions of a file. In a better world it would be possible for the source-code control system to handle all merges automatically, but this is not yet possible, so the act of performing a merge is really one of handling the exceptions to the merging process.

If the tool that calculates the differences between files in different branches is a poor one, then a larger amount of work will fall to you to integrate the differences. The reason people call it “merge hell” isn't because of the merging, per se, but because of the amount of human intervention that is required. Do you want the code in version 1.2.2.1 that Bob wrote to be merged with your code in your current version? These are the types of decisions that make merging so unpleasant. On a positive note, this is a known problem and many people are trying very hard to provide us with better tools, so it's one area in which things are actually improving.

To sum up, my advice is to integrate changes into your branch daily—but not at the beginning of the day—and to work with a modern source-code control system that has good support for merging changes and resolving differences.

KV

KODE VICIOUS, known to mere mortals as George V. Neville-Neil, works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, and rewriting your bad code (OK, maybe not

that last one). He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler who currently lives in New York City.

© 2009 ACM 1542-7730/09/1000 \$10.00