Divide and Conquer

THE USE AND LIMITS OF BISECTION

Dear KV,

Many of our newer developers—those who have worked only with git—seem to find bugs in their code only by using git's `bisect` command. This is troubling for a couple of reasons. The first is that often—once they find where the change occurred that caused the problem—they don't understand the cause, only that it happened between versions X and Y. The second is that they do not seem to understand the limits of debugging in this way, which, perhaps, is more a topic for you than for me to describe to you. Do you find this practice becoming more widespread and perhaps debilitating to good debugging?

Vivisected by Bisection

Dear Vivisected,

Nearly all new tools are both a blessing and a curse, as close readers of KV will know by now, and the ability to bisect a set of changes quickly is no different. It is quite definitely a blessing to have automation take over the tedious work of checking out a change, building the system, running a test, and seeing if the test fails, and then if it doesn't fail in the right way, doing this all over again until the change that introduced the bug is found. That kind of work is something you want automated, and, therefore, in that case it is a blessing—a limited one, but a blessing nonetheless. I mean, it's not manna from heaven, is it?

Tools such as bisection are great if, and only if, you have a well-understood bug that occurs with 100 percent consistency so that the bisection can work.

What you are asking me to rant about (you are asking for a rant, right?) is how such a tool can create lazy thinkers, and by extension, lazy engineers. Well, there are a few problems to talk about even before we get to whether having such automation leads to laziness.

Tools such as bisection are great if, and only if, you have a well-understood bug that occurs with 100 percent consistency so that the bisection can work. Bisection is of no use if you have a heisenbug, or something similarly subtle, that will fail only from time to time; and, while we do not want any bugs in our systems, we know that these subtle bugs are the hardest to fix and the ones that cause us—well, some of us—truly to think critically about what we are doing.

Timing bugs, bugs in distributed systems, and all the difficult problems we face in building increasingly complex software systems can't yet be addressed by simple bisection. It's often the case that it would take longer to write a usable bisection test (the damnable thing you must write to get the bisection to tell you where the bad change was) for a complex problem than it would to analyze the problem whilst at the tip of the tree.

Another thing that developers often fail to understand is that the bug may not be related to any previous change; it might be right there in front of them, staring back, in orange on black. I've watched several developers who were absolutely convinced that the bug was "somebody else's problem" run and rerun bisections only to realize that the actual problem was in their latest, uncommitted change. It is unfair to laugh at people in the middle of a

debugging session, and, with KV, it's a risk to life and limb, but it is still damned tempting.

What bisection provides all developers is simply another tool to find bugs in their code. Sure, the bug has to be easy to test for, likely can't be in a distributed system, and can't be a timing or a heisenbug, but it's still better than finding these simpler bugs by hand or writing your own script to do just what `bisect` is going to do.

Does this tool make us dumber? Probably not. What it does is allow a lower-common-denominator developer to find bugs; however, if that developer wishes to learn, the tool doesn't prevent that, and that's why such a tool is a boon to some and a cushion to others.

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. Neville-Neil is the co-author with Marshall Kirk McKusick and Robert N. M. Watson of* The Design and Implementation of the FreeBSD Operating System *(second edition). He is an avid bicyclist and traveler who currently lives in New York City.*