Aversion to Versions

RESOLVING CODE-DEPENDENCY ISSUES.

Dear KV,
Recently I've been trying to piece together a set of software packages that are supposedly intended to work together but seem very fragile. The main source of their fragility comes from how the developers "resolved" the dependencies between packages, libraries, and their own software. Their solution to making all the pieces work together was to encode the version of the library or package into the file system, as in, */opt/pkg-v2.87/lib/...*. As you might imagine, this causes no end of trouble for us consuming this software when a library or package is upgraded. I've counted no fewer than 30 locations where this was done. You cannot tell me that this is the right way to handle this particular problem, but these people are paid professionals, and we paid for their software. What would KV do?

Aversion to Versions

Dear Aversion,
There are many solutions to your problem, and only one of them calls for dissolving something that tastes like bitter almonds in the coffee pot of the corporate entity that sold you this software. If you do this and get caught, please don't tell anyone where you got the idea.

The problems of dependency analysis and resolution—as well as versioning—have been with us since the earliest days of the software library, and some of the solutions, such as SAT solvers for package systems, are clever and elegant and mostly work. Build dependencies are usually handled by systems such as `automake` and `autoconf`, which, so long as you never look inside them, are quite useful. If you look inside, you will not see how the sausage is made so much as how the animals were killed, diced, sliced, folded in triplicate, sold, bought, used as toilet paper, recycled into facial tissues, and finally spat back out as a makefile so long it will make your head spin—and not in that pleasant head-spinning way, but in a way where you have to lie down for an hour for the spins to go away. All of which is to say that these problems are solved, and the solutions are often complex and tortuous, but we all raise a glass—or a vial—to those who undertake to solve them.

Then there are those who, either through ignorance or stupidity, decide just to take a stab in the dark and solve the problem in their own, inimitable style. It is definitely these types you are dealing with today. I guess you could file a bug against the software and see if someone fixes it. But given the quality of what they have already given you, I think that's a long shot.

Since you've been able to count the number of these sins committed in software (and you number them at 30), I am assuming you have some amount of the source code; perhaps it was even all delivered as source. One quick and very dirty solution is to use the inimitable sed (stream editor) program to update the version numbers as necessary. A manual page can be found here, but I'm sure Stack Exchange or some other cheater site will give you code to "swap version numbers throughout my code" or some such thing. Just slap the code into a repo somewhere, find the right incantation of sed(1), sacrifice a live animal of your choice, and voilà, you'll be able to update the versions to match the latest library. Purists will scream that this is not the right way to solve the problem, and they're correct, but then purists rarely have a hot bar date they have to get to on a Friday night and thus have plenty of time to do the right thing, while the rest of us are trying to do the thing right.

The higher-level point here is that one should never hard-code a version or a path inside the code itself.

Of course, the better way—and again you'll need the source to do this—is to update the code to actually take a path argument or inquire after some sort of environment variable (MYLIBPATH) that can be used to point the software to the right place, no matter what version you want it to use. If you go this route, be sure to tell the developers that you'll send them the patch so long as they haven't already drunk the coffee you made for them.

The higher-level point here is that one should never hardcode a version or a path inside the code itself. Code needs to be flexible so that it can be installed anywhere (the hardcoding of /usr/local is blatantly foolish and yet persists) and run anywhere so long as the necessary dependencies can be resolved, either at build time for statically compiled code or at runtime for interpreted code or code with dynamically linked libraries. There are, as KV has just pointed out, current, good ways to get this right, so it's a shame that so many people continue to get it wrong.

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.*