DOI:10.1145/1897816.1897833

George V. Neville-Neil

Article development led by acmqueue
queue.acm.org

Kode Vicious
Forest for the Trees

Keeping your source trees in order.

Dear KV,

I've noticed that you comment a great deal on the cleanliness of people's code, comments, version numbers, and other coding habits, but you've never mentioned one of my pet peeves: people who can't seem to name their source trees correctly. Don't people who tell you, "Oh, that file is in ~my-name/project-foobar" annoy you? I can't imagine that they don't.

Frustrated by the Trees

Dear Frustrated,

There are so many things that frustrate me—as these columns have pretty clearly indicated—and so, yes, you are correct. People who don't store their checkouts neatly and in some reasonable fashion annoy me.

I often think that many programmers see their checkouts as they saw their rooms as children: a private domain in which they could do as they pleased until a parent told them to clean things up. With the amount of disk space available to the modern programmer, and the lack of parental supervision in most workplaces, the time to "clean your room!" never comes. Thus, their checkouts grow and accrete files they call temporary but that really should have been given a good home, or removed, long ago.

What happens next is that you're in a meeting or talking with said programmer and you ask, "Hey, where's the source data that you made that graph from?" or "Did you check in that useful script you wrote last month?" These people will invariably say, "Oh, I meant to, but it's just not that important. You can just go copy it from my tree. It's somewhere in my home directory under my-latest-work-17." "17" is their attempt at a version number, but don't expect them to have any directories labeled 1 to 16—really, just don't. Now you have to find the file, which you get to do via the excellent, and often slow, `find(1)` command. Hopefully they remembered the name of the file or you'll get to do multiple searches, which is never fun. The only thing that makes this kind of sloppiness worse is when it is done completely in public, in open source projects.

Most, if not all, open source projects allow you to follow them by using one

GENERATIVE ILLUSTRATION BY LEANDER HERZOG

of the current plethora of source-code control systems to check out their software to your local machine. While providing such a service is a great thing, providing it poorly is much like setting up a library in the middle of town, throwing all the books up in the air, letting them fall where they may, and then labeling some of them with Post-it Notes. Though most projects are not this horrific, I have noticed a tendency toward several sloppy, and therefore maddening, practices. I blame this trend on the recent introduction of distributed version-control systems, such as Mercurial and Git.

KV's first rule of public source-tree maintenance is to label everything clearly. Even if you don't think a tree will last very long, label it: give it a meaning that those who are new to your project can easily understand so they can figure out if that tree is, indeed, of interest to them.

My second rule is to not mix personal developer trees with release trees. A Web page with 100 different possible checkout targets—and you may laugh, but I see this on a regular basis—is not a good way to present your project to users; nor is it a good way to make code available. Keeping developer private source trees separate from trees you intend as real releases is a good way to increase sanity and reduce clutter. If people really need to check out a developer's private tree, they'll likely find it, though you might help them along by setting up a page labeled "Developer Trees."

And lastly, don't use developer trees as release trees. If the code in the developer's tree is good enough to make a release, then have the developer check it in, make a branch, and release it. A developer who is too lazy to do this should not be part of a project. No developer is important or brilliant enough for his or her tree to be the release tree.

KV

Dear KV,

In my spare time at work I've been adding an embedded language to some of our tools so that other people on my team could more easily script parts of their work. After spending a few weeks doing this, I showed what I had done to my team, and instead of them all being happy and welcoming the extra work, their reactions ran from indifferent to hostile. I even used a popular, open source, embeddable language, not something I cooked up on my own. I made their jobs easier. Why wouldn't they be happy?

Underappreciated

If the code in the developer's tree is good enough to make a release, then have the developer check it in, make a branch, and release it.

Dear Under,

Are you sure you made their jobs easier? Are you sure you understand their jobs? It is a common belief by engineers that every piece of code they write is somehow a boon to mankind and is helping to drive the entire human race forward, propelling us all into a brave new world. Another thing to consider is that most people do not like surprises, even good ones. Try this experiment. Take a $20 bill—or if you're in Europe a 10-euro note—and leap into a coworker's cubicle screaming, "Good morning!!!" at the top of your lungs and then loudly slap the bill on the desk. You've just given your coworker money, so surely that coworker will be happy to see you. Please report back your results.

What is more likely is that you found a need that you, yourself, wished to fill and you spent some enjoyable time filling that need. There *is* nothing wrong with working to scratch a technical itch; some of the best innovations come from engineers doing that. There is something wrong with believing that a group of people, who have no idea what you've been doing late at night for the past month, are suddenly going to look at whatever you've created and say, "Oh, joy! It's just what I wanted!" All but the most obvious of creations need to be socialized. (Yes, I used *socialized* in a technical column.)

If you want your idea to be accepted, you first have to understand whether it is needed by anyone except yourself. Doing this by secretly watching your coworkers and taking notes is a great way to get yourself put on some sort of psych watch list with HR, so I suggest you go about it a bit less subtly than that: by asking them. Ask one or two people you think would want to use your new software if they are actually interested in what you're thinking about building. If they say, "No," that's not a reason to stop; it just tells you that when you're done, you'll have to do a lot more work to get them to see how great your creation is. Just for the record, yelling at them in a meeting and telling them how stupid they are not to see how clever you are is also a losing strategy.

Your best bet is to think about a simple part of your new system that is so useful, and so incontrovertibly a boon to their daily lives, that they will immediately find a use for it. Concentrate on making that useful piece available to them, and you will likely win them over. Or, you could just become management and force them all to do your bidding. Either way.

KV

Related articles on queue.acm.org

Purpose-Built Languages
Mike Shapiro
http://queue.acm.org/detail.cfm?id=1508217

Broken Builds
Kode Vicious
http://queue.acm.org/detail.cfm?id=1740550

George V. Neville-Neil (kv@acm.org) is the proprietor of Neville-Neil Consulting and a member of the ACM *Queue* editorial board. He works on networking and operating systems code for fun and profit, teaches courses on various programming-related subjects, and encourages your comments, quips, and code snips pertaining to his *Communications* column.