Kode Vicious Strikes Again

Kall us krazy, but we're making Kode Vicious a regular. And so we say again, all together now: Never fear, Kode Vicious is here! Answering your questions, solving your problems, and making the world a better place.

Dear Kode Vicious,

I have this problem. I can never seem to find bits of code I know I wrote. This isn't so much work code—that's on our source server—but you know, those bits of test code I wrote last month, I can never find them. How do you deal with this?

Linguistically Lost

Dear Lost,

Losing things makes me feel stupid, and I hate feeling stupid. Several years ago a bright, but very spacey friend of mine pointed out an obvious answer to keeping track of all those things I have on my computers. Put them in a source repository! Not the work one, dummy! The last thing you want is to have your employer see all your personal code, documents, and stuff. I mean, what happens when they go under, as they so often do?

What I did was this: a few years ago I took one of my servers at home (you do have servers at home, yes?) and put a CVS (concurrent versions system) repository on it. I now keep everything I care about, which is just about everything, checked into my own personal CVS. This includes all of my configuration files (dot files since I'm a Unix head), all documents (including this letter and response), all code, all scripts, pretty much everything. Anything I create on a computer that I'm going to use more than once goes into the repository. Given the huge amounts of disk space available cheaply nowadays, it doesn't make any sense to throw away anything you think you might use again.

As a side benefit, I can now set up my environment on any computer in about 10 minutes. I just make sure CVS

Got a question for Kode Vicious? E-mail him at kv@acmqueue.com—if you dare! And if your letter appears in print, he may even send you a Queue t-shirt, if he's in the mood. And oh yeah, we edit letters for content, style, and for your own good!

A koder with attitude, KV ANSWERS

YOUR QUESTIONS.

MISS MANNERS HE AIN'T.

is on the new machine, and check out my Personal/DotFiles directory. Then I run a small script

that links all the necessary files from Personal/DotFiles to their correct places and, bang, instant KV koding machine. This can work equally well on non-Unix platforms, but I don't use those, so you'll have to work out the mechanics yourself.

Oh, and remember to back up the repository in some way. Mirror it, burn it to CD-ROM or DVD, but do something for that "just in case moment." If you think you feel dumb when you've lost one file, think of how absolutely, amazingly stupid you'll feel when you lose a few megabytes or more.

I'm sure the first letter I'll get after this column will be from someone whining about how to find things once you've been storing everything. I recommend you use DIRECTORIES, a feature of operating systems since the 1970s. I also suggest you use reasonable names that do NOT include "foo," "bar," and "baz"!

KV

Dear Readers,

In this issue, nasty old Uncle Vicious is not going to print and respond to a letter. Why is that? Well, it's not because y'all have been silent, by any means. This month I want to talk about my Top Five Coding Peeves. And why do I want to talk about this? Well, because I'm tired of seeing them in your code when I come to work where you've been working, that's why! So, in ascending order, here are the things that cause me to slug down unhealthy amounts of alcohol after work.

5. Crappy comments. You, yes you, you know who you are. You are the people who write comments like:

```
// Set i equal to 1
```

```
i = 1;
```

when that's obvious, then leave huge, complex functions, which do nine different things under 20 different cases, completely uncommented.

Oh, and how about remembering CS101 and placing a useful comment at the head of each function? Just

because you don't think anyone will see that function doesn't mean someone like me isn't going to have to clean up that mess later.

4. Dangling else clauses. One of the most common errors made when fixing or extending software is the addition of code inside what you thought was a block, which turns out not to be. This is what I call a dangling else, and I tend just to add braces when I see them. Why? Well, it's not like the two extra brace characters per else are going to overload your disk, unless you're programming on some very creaky hardware. In which case, ask your mother for \$50 to buy a bigger hard disk or delete some of those movies you've been downloading.

3. Magic numbers. I don't know how many times I've seen this:

```
name_buf[128]; // Hold the name of the file.
followed either by name_buf being incorrectly indexed or
the very same type of storage declared differently:
new_name_buf[127]; // Hold the name of the file.
```

No matter what language you use—C, Java, and so forth—at some point you're going to need to use constants and when you do, you should name them. Why? Well, not only is it a cleaner way to code, but it also provides much more consistent results since you're more likely to use the constant name everywhere you need it. Now, what if you port the code to a system where names are shorter or longer? Do you want to search all the code to find these magic numbers and change them?! I didn't think so.

This particular use of magic numbers is lamer than usual as there is a well-known, operating-system-defined constant that you should use. On the machine on which I'm writing this column, it's actually 1024 bytes and it's called `FILENAME_MAX`.

2. Code dingleberries. Nothing is more jarring when reading code than to come across a huge chunk of commented-out code in the middle of a file or function. Since you're using a source-code control system (wait, you are using a source-code control system, right?), you can simply remove the code and check in the new version. If you ever need the dead code to be resurrected, you can simply retrieve the older version.

The same complaint goes for blocks of code that are conditionally compiled out at build time using things like `#if/#endif`. There are some systems that depend on `#ifdef` for configuration, which I also find problematic, but this is not the same as the `#if 0` case. If you don't want the code to be in there, then don't put it in there, or make your source-code control system remember that it was once there. The code should always represent the system

as it is, not as it was, or should be.

1. Global variables. There is, I sincerely hope, a special level in hell reserved for those programmers who, every time they fail to make a proper abstraction, add another global variable to their programs. Above this, I hope there is yet another level, with even nastier punishments, for those programmers who give such global variables names like "s" and "r." Tracking down problems in software written like this is not impossible, but it's not my idea of fun. So please, if you really need a global variable, give it a name that is easy for others to find with `grep` or other tools like `Cscope`, and make sure you really, really, really need it.

So that's it, my top five pet peeves. Sorry, no prizes, not even a booby prize, just me screaming, "STOP DOING THAT!" I'll throw the floor open now, and let you write to me with your pet peeves. If I like them, I just might publish them. If I don't like them, well, there's always the "d" key.

KV

Dear Kode Vicious,

My boss keeps complaining that I have an NIH attitude and that all my best ideas are already "out there in the published literature," wherever that is. It's really getting me down. Have you thought of any clever ways to get him off my back?

Bummed

Dear Bummed,

Well, I do have my ways of keeping bosses off my back, but I'm not going to share them here. First because many are illegal, and second because I think your boss may be right.

It's been my experience that the last time many koders read a paper was when it was a requirement in some course at school. Although there are many books written about the latest buzzwords in the industry, these never cover the basics—the underpinnings of what we do every day—nor do they talk about major changes in how we build systems. It's all fine and good to learn the language of the week, but if you don't understand how and why languages are written, you'll probably pick the wrong one for your project. The same is true for any part of our field, whether it be databases, Web application frameworks, security products, or networking protocols.

"But journals are so boring!" I hear you whine. Well, yes, there isn't a lot of hot, bedtime reading out there, except for *Queue* of course. I will admit to taking a par-

ticularly good set of conference proceedings to bed, but, as you well know, I happen to be an extreme case.

What it comes down to is that it is your responsibility to be informed about your field. If you were a pipe fitter and you missed the switch from lead to copper, you would certainly be doing your customers and yourself a disservice.

There are a few hints I can give you here to make this process a lot easier. First of all you should ask around and see what your ko-workers read. Find people whose code or designs you respect and see what's on their shelves. Then borrow an issue or two from them, or read some of the articles online.

Once you have a few different journals head to a local coffee shop, or bring them on your next flight. Commuting is also an excellent time to read, but only if you're NOT DRIVING A CAR! There are two things you should remember when reading a journal. The first is that most, if not all, journals publish abstracts for every one of their papers. Read all the abstracts first, then decide which papers you want to read. The second thing to keep in mind is that papers are *not* novels. If you don't like the paper you're reading, and you're not getting anything

out of it, then put it aside. Life is too short to read poorly written papers.

Now, before I run out of space, I should put in a plug for the good folks here at ACM. This organization, of course, has journals and conference proceedings that cover every aspect of the IT industry, and that's a great place to start. My other two favorite sources of journals and proceedings are IEEE and Usenix. Each organization serves slightly different communities, but they're all worth a look. Now put down that copy of *Snow Crash* and get back to work!

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 has made San Francisco his home since 1990.

© 2004 ACM 1542-7730/04/1100\$5.00