GNL is Not Linux

WHAT'S IN A NAME?

Dear KV,

I keep seeing the terms *Linux* and *GNU/Linux* online when I'm reading about open-source software. The terms seem to be mixed up or confused a lot and generate a lot of angry mail and forum threads. When I use a Linux distro am I using Linux or GNU? Does it matter?

What's in a Name?

who is KV?

click for video

Dear Name,

What, indeed, is in a name? As you've already seen, this quasi-technical topic continues to cause a bit of heat in the software community, particularly in the open-source world. You can find the narrative from the GNU side by clicking on the link provided in the postscript to this article, but KV finds that narrative lacking, and so, against my better judgment about pigs and dancing, I will weigh in with a few comments.

If you want the real back story on the GNU folks and FSF (Free Software Foundation), let me suggest you read Steven Levy's *Hackers*, which is still my favorite book about that period in the history of computing, covering the rise of the minicomputer in the 1960s through the rise of the early microcomputers in the '70s and early '80s. Before we get to the modern day and answer your question, however, we have to step back in time (cue bad time-travel effects) to the late 1960s, and the advent of the minicomputer, which came along,

strangely enough, at the same time as the miniskirt.

One upon a time, as all good stories start, nearly all computer software cost some amount of money and was licensed to various entities for use. That time was the 1950s and 1960s, when, in reality, very few individuals could afford a computer, and, in fact, the idea that anyone would want one was scoffed at by the companies who made them, to their later detriment. Software was developed either by the hardware manufacturer to make its very expensive machines even moderately usable, or by the government, often in collaboration with universities.

About the time of the advent of the minicomputer—which came along about the same time KV was born, screaming, because he knew he would have to fix them and their brethren someday—two key innovations occurred. Ken Thompson and Dennis Ritchie invented Unix, a now well-known reaction to the development of Multics, and computer hardware took one of its first steps toward affordability. No longer would the whole university have to share and time-slice a single large mainframe; now each department, if that department had $30,000 or so, could share a less powerful machine, but among a much smaller group of people.

Before Unix, operating systems were large, complicated, and mostly nonportable. Because Unix was simple and written in a new, portable assembler, which we now call C, it was possible for much smaller groups of people to write significant software with a lot less effort. There were good tools for writing portable operating systems and systems software, including a compiler, linker, assembler,

and debugger—tools we now take for granted. All of these advances were significant and important, but they had one thing holding them back from even broader acceptance: licensing.

The Unix system and its tools were written and owned by AT&T, which, at that time, was the largest monopoly in the United States. It didn't have to license anything, of course, because it was *the* phone company, and *the only* phone company, which put it in a unique position, best summed up by Lily Tomlin in an old comedy sketch: "We don't care, we don't have to."

The truth was that many people inside AT&T did care and they were able to get AT&T to sell "cheap" licenses for institutions such as universities that had to pay only $1,000 to get the entire source. Companies had to pay quite a bit more, but they were able to spread the cost over their entire computing infrastructure. Having the source meant you could update and modify the operating system. If you want to see innovation in any type of software, it's very important to have the source. In 2015, 50 years after all these changes started, it's now common to have access to the source, because of the open-source movement, but this was uncommon at the start of the Unix age.

Over the course of the Unix era, several open-source operating systems came to the fore. One was BSD (Berkeley Software Distribution), built by CSRG (Computer Software Research Group) at UC Berkeley. The Berkeley group had started out as a licensee of the AT&T source, and had, early

on, written new tools for AT&T's version of Unix. Over time CSRG began to swap out parts of the system in favor of its own pieces, notably the file system and virtual memory, and was the first to add the TCP/IP protocols, giving the world the first Internet (really DARPA net)-capable Unix system.

At about the same time, FSF had, supposedly, been developing its own operating system (Hurd), as well as a C compiler, linker, assembler, debugger, and editor. The effort to build tools worked out better for FSF than its effort to build an operating system, and, in fact, I've never seen a running version of Hurd, though I suspect this missive will generate an email or two pointing to a sad set of neglected files. The GNU tools were, in a way, an advancement, because now software developers could have an open-source set of tools with which to build both new tools and systems. I say, "in a way," because these tools came with two significant downsides. To understand the first downside, you should find a friend who works on compilers and ask if he or she has ever looked inside gcc (GNU C compiler), and, after the crying stops and you've bucked your friend up with more alcohol, ask if he or she has ever tried to extend the compiler. If you are still friends at that point, your final question should be about submitting patches upstream into this supposedly open-source project.

The second downside was religious: the GPL (GNU Public License). If you read *Hackers,* it becomes quite obvious why FSF created the GPL, and the copyleft before it. The people who created FSF felt cheated when others took the software they had worked on—and which was developed under various government grants—started companies, and tried

to make money with it. The open-source community is very clearly split over the purity of what one develops. There are those who believe that no one should be able to charge for software or to close it off from others, and those who want to share their knowledge, whether or not the receiver of that knowledge makes a buck with it.

All of this background brings us to Linux and its relationship to the GNU tools. Linux is an operating-system kernel, initially developed by Linus Torvalds in reaction to the Minix operating system from Andrew Tanenbaum. Torvalds used the GNU tools—compiler, linker, assembler—to turn his C code into an operating-system kernel and then launched it upon the world. He released the code under a GPLv2 license, the one it maintains to this day, rather than taking on GPLv3, which is even more restrictive than its predecessors. Other people took up the code, modified it, improved it, and built new tools and systems around it.

We don't name things after the tools we use to build them; we name things in ways that make sense because they describe the whole of the thing clearly and completely.

Now, to the point about naming. When you build a house, you use many tools: hammers, saws, drills, etc. When the house is complete, do you call that a Craftsman/House, a Makita/House, or a Home Depot/House? Of course you don't, because that would be stupid. We don't name things after the tools we use to build them; we name things in ways that make sense because they describe the whole of the thing clearly and completely. Linux is an operating-system kernel and some associated libraries that present a mostly, but not always, Posix, Unix-like system on top of which people write software. In point of fact, Linux distributions do not ship with the GNU tools, which have to be installed from packages later. Linux

is a thing unto itself, and the GNU tools are things unto themselves. That one might be used to work on the other is irrelevant, as is the fact that I'm holding a Craftsman hammer from Home Depot right now... must put the hammer down.

The whole GNU/Linux naming silliness is probably a case of kernel envy, not unlike the programmers who feel that their ultimate achievement is to write a process or thread scheduler, which I addressed in "Bugs and Bragging Rights" (http://queue.acm.org/detail.cfm?id=2542663), where I wrote, "I think the propensity for programmers to label their larger creations as operating systems comes from the need to secure bragging rights. Programmers never stop comparing their code with the code of their peers."

Why is this important? There are two reasons. One is intellectual honesty. KV prefers to see the credit go to those who did the work. Linus Torvalds and his team have built an important artifact, out of many tools, that many people use each day, so the credit goes to them, not to the producers of the tools. It takes more than a compiler, linker, and editor to build something as complex as an operating system, or even the operating-system kernel, and many of the tools that go into building Linux have nothing at all to do with FSF or GNU. Should we now rename all of our systems as GNU/APACHE/FOO/BAR? Only a lawyer would think of that, and by now you all know what I think of letting lawyers name projects. The second reason this is important is to point out that while GNU stands for "GNU is not Unix," that was a reaction against the AT&T Unix of the 1980s. Now it might as well be "GNU is not

Linux," because the tool is not the thing the tool builds. But then, GNL doesn't sound as good.

KV

PS If you want to read the GNU side of this story, pour yourself a strong beverage and start here: http://www.gnu.org/gnu/linux-and-gnu.html.

LOVE IT, HATE IT? LET US KNOW feedback@queue.acm.org

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.*