Linux at 25: Q&A With Linus Torvalds

By **Stephen Cass**, spectrum.ieee.org

Linus Torvalds created the original core of the Linux operating system in 1991 as a computer science student at the University of Helsinki in Finland. Linux rapidly grew into a full-featured operating system that can now be found running smartphones, servers, and all kinds of gadgets. In this e-mail interview, Torvalds reflects on the last quarter century and what the next 25 years might bring.

Stephen Cass: You're a much more experienced programmer now versus 25 years ago. What's one thing you know now that you wish your younger self knew?

Linus Torvalds: Actually, I credit the fact that I *didn't* know what the hell I was setting myself up for for a lot of the success of Linux. If I had known what I know today when I started, I would never have had the chutzpah to start writing my own operating system: You need a certain amount of naïveté to think that you can do it. I really think that was needed for the project to get started and to succeed. The lack of understanding about the eventual scope of the project helped, but so did getting into it without a lot of preconceived notions of where it should go.

Read the feature, *Linux at 25*
The fact that I didn't really know where it would end up meant that I was perhaps more open to outside suggestions and influence than I would have been if I had a very good idea of what I wanted to accomplish. That openness to outside influences I think made it much easier, and much more interesting, for others to join the project. People didn't have to sign on to somebody else's vision, but could join with their own vision of where things should go. I think that helped motivate lots of people.

S.C.: Is there one early technical decision made during Linux's development that you now wish had gone a different way?

L.T.: The thing about bad technical decisions is that you can always undo them. Yes, it can be very frustrating, and obviously there's all the wasted time and effort, but at the same time even that is not usually really wasted in the end: There was some reason you took a wrong turn, and realizing that it was wrong taught you something. I'm not saying it's really a good thing—it's obviously better to always make the right decision every time—but at the same time I'm not particularly worried making a choice. I'd rather make a decision that turns out to be wrong later than waffle about possible alternatives for too long.

We had a famously bad situation in the Linux virtual memory subsystem back in 2001 or so. It was a huge pain, and there was violent disagreement about which direction to take, and we had huge problems with certain memory configurations. Big swatches of the system got entirely ripped out in the middle of what was supposed to be a “stable” period, and people were not happy.

But looking back at it, it all worked out in the end. It was painful as hell at the time, and it would have been much nicer to not have had to make that kind of big change mid-development, but it wasn’t catastrophic.

S.C.: As Linux grew rapidly, what was the transition from a solo to an ensemble effort like on a personal level?

L.T.: There really were two notable transitions for me: One fairly early on (1992), which was when I started taking other developers’ patches without always rewriting them myself. And one much later when [applying all the patches myself] was starting to be a big pain point, and I had to learn to really trust all the various submaintainers.

The first step was the much easier one—since roughly the first six months of Linux kernel programming had been an entirely solo exercise, when people started sending me patches I just wasn’t really used to just applying them. So what happened is that I would look at the patch to see what the person was aiming for, and then I would just do that myself —sometimes very similarly, sometimes in a totally different way.

But that quickly became untenable. After a fairly short while I started to just trust certain people enough that instead of writing my own version of their idea, I’d just apply their patch. I still ended up making changes often, and over the years I got really good at reading and editing patches to the point where I could pretty much do it in my sleep. And that model really worked well for many years.

But exactly *because* the “apply other people’s patches” model worked so well for years, and I got very used to it, it was much more painful to change. Around 2000 we had a huge growth in kernel development (at that point Linux was starting to be a noticeable commercial player). People really started to complain about my workflow being a roadblock to development, and complaining that “Linus doesn’t scale.” But we had no good tools to handle source management.

That all eventually led up to the adoption of BitKeeper as a source code maintenance tool. People remember BitKeeper for the licensing brouhaha a few years later, but it was definitely the right tool for the job, and it taught me (and at least parts of the kernel community) about how source control *could* work, and how we could work together with a

more distributed development model where I wasn't the lone synchronization point.

Of course, what I learned about how to do distributed-source-control management is how Git came about in 2005. And Git has obviously become one of the big success stories in source control, but it took a *lot* of teaching others about the advantages to distributed source control. The pain that the kernel went through in around 2000 was ultimately a big learning lesson, but it was unquestionably painful.

S.C.: Are there any other projects, as with distributed source control, that are giving you an itch you'd like to scratch?

L.T.: No. And I really hope there won't be any. All my big projects have come from "Damn, nobody else did this for me" moments. I'm actually much happier when somebody else solves a problem for me, so that I don't have to spend a lot of effort doing it myself. I'd much rather sit at a beach, sipping some frou-frou drink with an umbrella, than have to solve my own problems.

Okay, I'm lying. I'd be bored after a few days. I'm really happy that I have Linux because it's still interesting and intellectually stimulating. But at the same time it definitely *is* true that starting new projects is a very frustrating endeavor.

S.C.: Why do you think Linux never became a significant presence on mainstream desktops?

L.T.: Hey, still working on it. And I think Chromebooks are actually doing reasonably well, even if it's a fairly limited desktop environment, and not the full traditional Linux workstation model.

As to *why* the desktop is such a hard nut to crack—there are multiple reasons, but one of the big ones is simply user inertia. The desktop is simply unique in the computing world in that it's both very personal—you interact with it rather intimately every day if you work with computers—but also complicated in ways many other computing environments aren't.

Look at your smartphone. That's also a fairly intimate piece of computing technology, and one that people get pretty attached to (and one where Linux, thanks to Android, is doing fairly well). The desktop is in many ways more complex, with much more legacy baggage. It's a hard market to enter. Even more so than with a cellphone, people really have a certain set of applications and workflows that they are used to, and most people will never end up switching operating systems—the number of people who install a different OS than the one that came preinstalled with the machine is pretty low.

At the same time, I think it's an important market, even if to some degree the whole "general-purpose desktop" seems to be fading, with more specialized, and thus simpler, platforms taking on many tasks—smartphones, tablets, and Chromebooks all being examples of things that aren't really meant to be fully fledged general-purpose environments.

S.C.: What use of Linux most surprised you?

L.T.: These days? Not that much, since I think Linux has almost become the default environment for prototyping new hardware or services. If you have some odd, specialized device or if you're creating some new Internet infrastructure or whatever, I'm almost surprised when it *doesn't* run Linux.

But those "oddball" use areas *used* to surprise me, back when I still thought of Linux as a workstation and server operating system. Some of the early commercial Linux conferences when people started showing off things like gas pumps or fridges that ran Linux—I was blown away. When the first TiVo came out, the fact that it was running Linux was as interesting as the whole "you can rewind live TV" thing was.

S.C.: What's the biggest challenge currently facing Linux?

L.T.: The kernel is actually doing very well. People continue to worry about things getting too complicated for people to understand and fix bugs. It's certainly an understandable worry. But at the same time, we have a lot of smart people involved. The fact that the system *has* grown so big and complicated and so many people depend on it has forced us to have a lot of processes in place. It can be very challenging to get big and have invasive changes accepted, so I wouldn't call it one big happy place, but I think kernel development is *working*. Many other open-source projects would kill to have the kinds of resources we have.

That said, one continual challenge we've always had in the kernel is the plethora of hardware out there. We support a lot of different hardware—almost certainly more than any other operating system out there, but there's new hardware coming out daily. The embedded area in particular tends to have hardware platform development time frames that are often very short (you can pretty much turn around and create a new phone platform in China in a month or two), and trying to work together in that kind of environment is tough. The good news is that a lot of hardware manufacturers are helping. That didn't used to be true.

S.C.: What current technical trends are you enthusiastic about? Are there any that dismay you?

L.T.: I have always been interested in new core hardware, particularly CPUs. That's why I started doing my own OS in the first place, and I'm still excited to see new platforms. Of course, most of the time it's fairly small tweaks on existing hardware (and I very much believe that that is how technical development should happen), but it's still the kind of thing I tend to try to keep track of.

In a bigger picture, but not an area I personally get involved with, it's very interesting to see how AI is finally starting to really happen. AI used to be one of those "it's two decades away" things, and it would *stay* two decades ahead. And I was very unimpressed with all the rule-based models that people used to do.

Now, finally, neural networks are starting to really come into their own. I find that very interesting. It's not an area I work in, and not really something I foresee working on, but it's still exciting. And unlike the crazy LISP and Prolog language approaches, recurrent neural networks we know work from nature. And no, I'm not dismayed by the fact that true AI may finally start to be happening, like clearly some people are. Not at all.

S.C.: Do you think Linux will still be under active development on its 50th anniversary? What is the dream for what that operating system would look like?

L.T.: I'm not a big visionary. I'm a very plodding pedestrian engineer, and I try to keep my eyes firmly on the ground. I'll let others make the big predictions about where we'll be in 5, 10 or 25 years—I think we'll do fine as long as we keep track of all the small day-to-day details, and try to do the best we can.

It might be more interesting if the world was about big revolutions and how things would look radically different 25 years from now. But many of the basic issues with operating systems are the same today as they were back in the sixties, when people started having real operating systems, long before Linux. I suspect that we've seen many more changes in how computers work in the last 50 years than we're necessarily going to see in the future. Hardware people, like software developers, have simply learned what works and what does not.

Of course, neural networks, et cetera, will change the world, but part of the point with them is that you don't "program" them. They learn. They are fuzzy. I can pretty much guarantee that they won't *replace* the traditional computing model for that very reason. People will want smarter machines, but people will also want machines that do exactly what they're told. So our current style of "old-fashioned" computing won't be going away; it'll just get augmented.