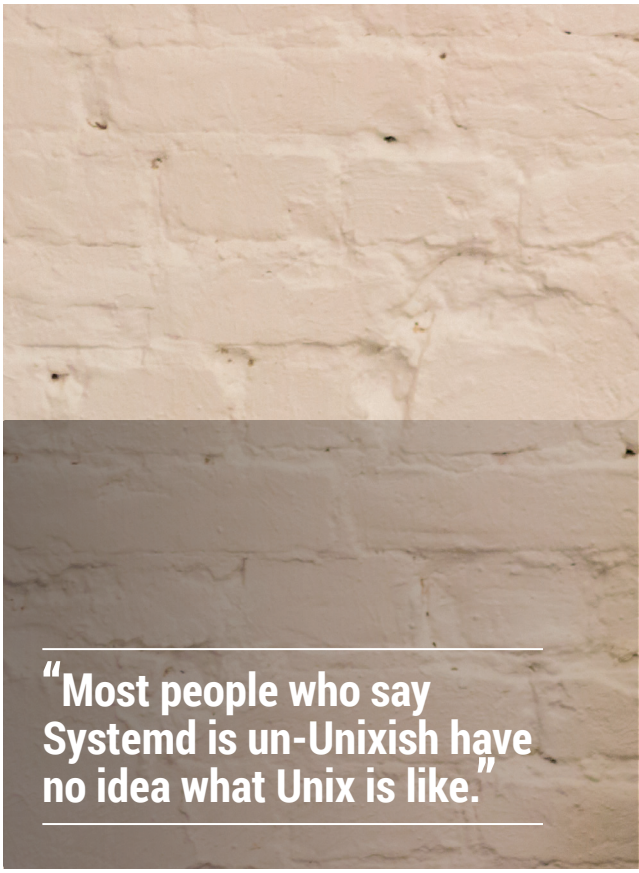
REBUILDING LINUX FROM THE GROUND UP

We meet **Lennart Poettering**, the lead developer of Systemd, an ambitious (and controversial) attempt to rewrite the Linux base system.

Few pieces of software in history have been so fiercely debated as Systemd. Initially a replacement for Sysvinit, the boot scripts that start up a Linux installation, Systemd has grown into a hugely powerful – and sometimes complex – replacement for the "bag of bits" that make up the Linux base system. It's growing all the time and now handles logging, device hotplugging events, networking, scheduled actions (like Cron) and much more. Almost every major Linux distribution has adopted Systemd, but there are still some unhappy campers out there, so Mike and Graham ventured to Berlin to meet Systemd's lead developer and get his view.

We also looked beyond Systemd, and Lennart's intriguing proposals for a new packaging system to make life easier for third-party application developers...



Systemd has now been adopted by pretty much every major distribution, and yet whenever it gets mentioned in discussions on the web, flamewars erupt. What do you think are still the biggest misconceptions?

Lennart Poettering: There are many different misconceptions. Something you always see is the claim that Systemd is monolithic – and another is that it's not like Unix. The Unix misconception is a pretty interesting one, because most people who say Systemd is un-Unixish have no idea what Unix is actually like.

What's typical for Unix, for example, is that all the tools, the C library, the kernel, are all maintained in the same repository, right? And they're released in sync, have the same coding style, the same build infrastructure, the same release cycles – everything's the same. So you get the entire central part of the operating system like that. If people claim that, because we stick a lot of things into the Systemd repository, then it's un-Unixish, then it's absolutely the opposite. It's more Unix-ish than Linux ever was!

The Linux model is the one where you have everything split up, and have different maintainers, different coding styles, different release cycles, different maintenance statuses. Much of the Linux userspace used to be pretty badly maintained, if at all. You had completely different styles, the commands worked differently – in the most superficial level, some used **-h** for help, and others **--h**. It's not uniform.

If we put a lot of the glue in one repository, it's not all the way towards Unix, but it's half way between traditional Linux and traditional Unix. We do not put libc and the kernel in the same repository, just the basic things. So that's a misconception that I'm always bemused about, and I'm pretty sure that most people who claim that have never actually played around with Unix at all.

Another issue is: some people see Systemd presented as an init system replacement, but now it's doing X, Y and Z on top. You've said it's about replacing a "bag of bits" with an integrated suite of tools. When you started Systemd, was it a case of Red Hat saying to you, "we want a new init system", or...

LP: No, it was actually the opposite. Back in the day, when we started working on Systemd, many of us who worked on the lower levels of the operating system realised that Sysvinit was not going to be the future. And then I was playing around with writing my own init system, which had the funny name Babykit, and that was 10 years ago or something. And then Canonical's Scott James Remnant started working on a new init system called Upstart. He made it public, and I stopped working on Babykit.

We, at that time, thought: OK, Upstart is the future! Scott understood how init systems work – it needs to be dynamic, it needs to react to events, and it's not the static thing that Sysvinit was. So we thought that was the way of the future, but as it progressed, we realised it probably wasn't the future, because we realised that conceptually, it was the wrong design.

The way Upstart worked is that, as a programmer or admin, you write: if A happens to B, or X happens to Y, do a certain thing. But we believed that an init system should work the other way around, where you say: this is where I want to go to, and you figure out the rest. Because of that design, Upstart was very simple, but it put a lot of complexity on admins and developers,

because you actually had to write down all these rules. It wasn't the computer that figured out what to do.

We thought: if you want to solve this properly, then you need to let the computer do these things. And this had lots of different effects: for example, Upstart always maximised what happened on the system, while we think you always have to minimise what happens. And the reason for that was simply because, if you specify exactly what state you want to end up in, you can pull in all the dependencies recursively and boot to exactly that.

"We started writing Systemd, and Red Hat didn't like it at all. So I worked in my free time."

The Upstart way is always, "if this is started, then start that". If the network is up, you take that as a trigger to start NFS and things like that. It always has this effect that you start as much as possible instead of as little as possible.

So anyway, long story short, we came to the conclusion that Upstart is conceptually wrong, and it moved at glacial speeds. It also had the problem that Canonical tried very hard to stay in control of it. They made sure, with copyright assignment, that they made it really hard to contribute, but that's what Linux actually lives off. You get these drive-by patches, as I would call them, where people see that something is broken, or something could be improved. They do a Git checkout, do one change, send you it and forget about it.

LV And you never see them again!

LP: Yeah, and this is great – these are the people you want to have, because the vast majority of patches are actually of that kind. It gives you this polishing that you want. The people invested in the project all the time do the big things, and don't care so much about the polishing. So these kind of patches are what you want. But if you do these copyright assignment things, you will never get those people because they would have to sign a contract before they can send you something.

Putting it all together, we realised that Upstart wouldn't be it. So at one Linux Plumbers Conference, four years ago or so, Kay Sievers and I said that we should do something about it, after we saw at the conference how Upstart wasn't moving ahead. And then we started working on it, pulled out the old Babykit code, gave it a new name, and started proposing it.

A lot of people understood that this was the better approach. It was a lot more complex than Upstart – to make it clear, I think Upstart actually has its benefits. The source code is very, very nice, and it's very simple, but I think it's too simple. It doesn't have this engine that can figure out what the computer is supposed to be doing.

So we started writing Systemd, and Red Hat didn't like it at all. Red Hat management said: no, we're going for Upstart, don't work on that. So I said, OK, I'll work on it in my free time. Eventually Red Hat realised that the problems we solved with Systemd were relevant, and were problems that needed to be solved, and that you couldn't ignore them.

Then we convinced the Fedora Technical Committee to adopt it, and then Red Hat internal management accepted it for RHEL, and we managed to convince every committee that mattered, bit by bit. It was absolutely not that Red Hat told us to work on it – we had to convince them.

LV I don't think many people know that!

LP: This is something that people in general don't know. They assume that Red Hat is this one entity, that has one opinion and pushes one thing. It's really not like that. The people who work at Red Hat, the engineers, they come from the community – they first become famous in the community, they hack on things, do good stuff, and then Red Hat comes along and says, "Hey, do you want to work for us?"

And when you start working for Red Hat, they don't check your opinions at the door. You can be sure that if there are multiple opinions on one topic in the broader community, the very same opinions inside Red Hat exist too. Inside of Red Hat there are discussions. Red Hat has many different people, and most of them have strong opinions and convictions.

LV And much of this debate happens in public, on public mailing lists. Then you have some people saying that all this arguing

looks bad, compared to how Microsoft or Apple does things. But I bet they all have the same arguments, just as passionately.

LP: I'm absolutely sure. There was this time when the people working on *Microsoft Word* had their own compiler to build *Word* and the rest of *MS Office* with. Microsoft had the Visual Studio group, and the Office group, and they had their own individual compilers. That's just crazy of course.

So I don't think that Red Hat is different from anywhere else; except that at Red Hat, because people are working on open source, they have much greater attachment to their code. So they have even stronger opinions.

LV If back at the start of Systemd, you and the other developers had explicitly said: "We're going to replace a lot of the base system", do you think it would've been better received? Some people see it as an init system that's suddenly touching everything else.

LP: Initially it was an init system – it was just PID 1. We knew from the very beginning what we were getting ourselves into. We knew very well that touching something that has so much history, that is so close to what admins do all day... That changing it would be a massive problem. So we knew that people would hate us for it. We knew we'd have to fight for a long time to get it accepted.

We eventually realised that doing just the init system would never be a complete solution. Because if you do an init system but still invoke all the shell scripts and all the other things needed to bring up the system, you've only solved part of the problem. You've solved one thing but not 90% of the problem. So we slowly started doing stuff that all the other Linux distros did, and implemented that in simple C code that was fast and parallelised.

Debian had its init scripts, and Fedora had its init scripts, and they all kind of did the same thing, and did it differently, and some are better, and some are worse. We thought OK: this is bullshit, let's write this in C in a unified way, and try to pick the best features of all distributions and make a convincing argument that it's the right way.

So it initially grew. But something to realise is that there's very little in Systemd that's actually required. Systemd requires Journald, because every single service that runs on the system is connected to Journald, and we need some way to log things during early boot. So Journald is a requirement, and Udev is a requirement. But pretty much all other components are completely optional. If you don't want to use the way Systemd loads kernel modules from a static list, then you can absolutely replace it.

Or if you don't want to use some of the more modern components like Networkd, then use something else. I mean, on my laptop I even use NetworkManager, because Networkd doesn't do wireless, right? Networkd is more for containers and servers. So if you want to adopt Systemd, you can absolutely adopt the baseline, which is the three components that I mentioned. You can keep the rest of the system – however, our implementation of the individual parts is usually pretty convincing, and usually people then replace more.

> **"Most people at Canonical didn't even realise that they had commit access Systemd."**

LV Some people see it as a requirement for Gnome...

LP: But it's not actually a requirement. Some people don't realise that when Gnome started making use of Logind, I actually wrote the patch for that. I ported *GDM* onto Logind. But when it did that, I was very careful to make sure it would still run on ConsoleKit. I didn't

You need a thick skin to hack in open source code sometimes, especially if half of the world seems against you.

Lennart lives in Berlin, and knows where to get great Vietnamese grub.

want to have those fights – if people want to continue running ConsoleKit, they can. Those patches made it in, but some people saw that Gnome now works with Logind, hence it must not work with ConsoleKit any more!

But that's actually not true. And to my knowledge the code is still in there – the compatibility for ConsoleKit. The Gnome team has the general problem though, that nobody's willing to maintain it. People who want to stick to the old stuff, they actually need to do some work on it. If they don't, then it will bit-rot and go away.

So anyway, we tried to do these things in the nicest possible way, but of course people generally don't acknowledge it!

A lot of people just think there's only Red Hat working on Systemd.

LP: Oh yeah, we're a lot of people now. Yesterday we had 26 committers, and 40 people contributing code every month or so. The committers group is quite diverse, and for us it's quite an exercise in making the diversity of the community be reflected in the diversity of the people who work on it. This is also related to how Upstart worked: Upstart was very locked-down, and Canonical always wanted to stay in control of everything. For us it was an exercise to make sure this doesn't happen. We're not the ones in power – the community is.

So of those 26 committers, there's a good chunk working for Red Hat right now, but there are people from Intel, Canonical... We had people from Canonical in the committers group, all the time during the discussion about whether they should even adopt Systemd. Most people at Canonical didn't even realise that they had commit access to these things.

There are also developers from Debian – two or three of them.

There should be a Systemd foundation!

LP: [Laughs] Well, we don't want to make it too formal. We have this speed, this quick pace with how we progress Systemd, and I think it can only work if we stay somewhat loose and not have strict regulations about how these things work.

But we try to make sure that it's inclusive. We have people from Arch Linux, people from all the Linux distributions, big companies that do open source. We want to make sure it stays that way.

Why do you think some distributions managed to adopt Systemd without any major fights, and then others like Debian had very intense debates and resignations? Is it just because it's a distro with more political processes?

LP: Arch Linux probably did it the quickest way. You know, distributions attract different kinds of people, of course. If you looked at Arch Linux, it attracted very progressive kinds of people – like power users. They're progressive and want to make the best out of their computers. So it was easy for them to adopt.

Then if you look at Gentoo, for example, they still haven't done Systemd as default. They used to be like Arch Linux is now – they used to be the young people who adopted things quickly. But the Gentoo people aged, and they became more conservative.

And Debian is probably an even more conservative bunch. Debian is a really old project, and many people from back in the old days are still active on it. So they have longer release cycles. And Fedora always defined itself as being on the bleeding edge, of course, so it was easier. Well, not that easy – some people don't realise that inside of Fedora and inside of Red Hat, there

After Avahi, PulseAudio and Systemd, we're intrigued to see what Lennart tackles next…

were lots of fights. So it's to do with the culture around the various distributions. And Slackware are the ultra conservatives!

Do you read the comments when Systemd is being discussed on the net? Do you despair when it all turns into hatred and flamewars?

LP: For some reason it doesn't touch me too much. I try to keep an open mind and figure out what people actually think. There's a lot of noise out there, but usually there's some core of an argument – something that we should actually be aware of. So if people are annoyed by Systemd, usually they ran into some kind of bug or something. It might not necessarily be a Systemd bug, but we need to take it seriously.

Nowadays Systemd is very polished in many ways, and the reason why it is so polished is because we actually listen to people. Sometimes people say we don't listen – we do, but we just don't always agree. If we would just stick our heads in the sand and not care at all what people wrote, Systemd would certainly not be what it is, or have found the adoption that it has.

You've said yourself that the flamewars could dissuade potential open source contributors from getting involved.

LP: I'm in the lucky position in that there's no pressure on me in any way. I know that a lot of people have pressures that they live under, and if you also get pressure from the internet over some things that you do in your free time, because you love it, that is very disappointing for them. So I have a luxury, and I know it, and I can only feel for people where it's not like that.

I know a lot of people who've had enough of open source, and who will not participate in the communities where things get really bad. And that's a big loss for open source.

Something else we wanted to talk about is your proposal for packaging. What's that about?

LP: It's really about augmenting the Linux platform with a new way to package applications. It's not about simplifying things or changing things – it's adding something to the ecosystem that we were missing so far. There are lots of people working on that in different areas.

If you look at all the operating systems that are popular these days, like Android, Mac OS, iOS, Windows Metro, they always have really strong app platforms, where they provide a sandbox, a nice way to distribute apps and ship updates. On Linux we don't have anything that's as convincing. We don't have a common way to sandbox stuff, and the way that we ship stuff is with Deb packages, or RPMs.

It's madness for third-party app developers, to develop for Linux. Like, what do they develop against – which distribution? And if they make that decision about which distributions they wanted to support, it'd be quite a few, and then there are lots of versions. You might want to support Fedora 20, 21, and 22, and then OpenSUSE and its various versions, and Ubuntu and its various versions… All of those distributions bring different libraries.

So the test matrix, the combinations of software you have to test your apps with, grows incredibly. That's not something that's digestible for third-party app developers. And it's really hard. The only way you can really deal with that is to get your stuff into the distributions. Then the distributions will do all the work for you – they'll rebuild for you, test for you and things like that.

But for upstream developers it's hard to get new releases into distributions quickly.

LP: Exactly – you're bound to what the distributors do, to the lifecycles of the distributions, the release cycles. You're

not responsible any more for your software – you've passed it on to the distributions. Which in many cases is actually a good thing, but in general it's not what third-party developers want. If you look at how *Firefox*, for example, packages its Linux version, it's a tarball that installs in its own directory.

The classic Linux distribution model, where you get everything from the distribution nicely packaged up, vetted for security problems, with security updates – that's a fantastic thing. But I also think it leaves out all these third-party people, and if we want to grow the Linux ecosystem beyond what we already have, into something where it's actually useful for a broad number of people, where we have markets and more apps, we need to provide a way to make these apps digestible.

And by digestible I mean: we need to have good sandboxes. If you don't get your software from the distribution any more but directly from the developers of the software, then you have the problem that you can't trust the code as much. Distributions add a bit of trust, in that they look at the code before packaging it, and then you get a nice stamp on it: this is good software. And then you only have to trust the distribution, and not trust 100 different software vendors any more.

Now, if we open this up and make it typical that you install one distribution and then 100 different apps from 100 different vendors, we need to do something about this trust problem. So that's why we need sandboxing. We need to reduce the chance that badly behaving software can destroy your data.

OK, but how do you deal with the niggling little differences between distributions, where everything has slightly different library versions, filesystem locations?

LP: Our idea is to introduce something called runtimes. When third-party developers develop their stuff, they should be able to do so against one very fixed set of libraries, in very specific versions, compiled in a very specific way. They can test their stuff with that – and then, if they're sure that everything works, they can check it off, and when it's installed on the final machine, it knows exactly the combination of software that it runs against. Instead of some weird combination of software local to that machine, where they have real trouble making sure it works.

Third-party developers don't want to do all the support for someone who says: "Yeah, but I have this old version of libc and it doesn't work", or "I have this really custom distro I compiled myself". So we have this concept we call a runtime, which is basically just a set of libraries with very specific versions. The idea is that you can install multiple runtimes at the same time. And then, if you have apps that require different runtimes with different versions, they'll run against their specific runtime and everything's good.

"Sometimes people say we don't listen – we do, we just don't always agree."

After our interview, Lennart pointed us in the direction of the East Side Gallery for some Berlin Wall exploring fun.

Didn't the Linux Standards Base try to do something similar for third-party devs?

LP: What the LSB did there was to standardise a set of libraries, but didn't define any specific versions. And classic Linux distributions only allowed installation of one set of libraries at a time – you could have one libc, and one OpenSSL. So LSB tried to make the best out of the traditional Linux model, but that's not enough.

There's a scheme that we put together that's not unlike what Android has. For example, if you develop an Android app, you do so focusing on one specific runtime, right? It's one that Google defines, and if there's something that's not in this huge runtime, then you have to ship it yourself inside of the app, and everything is good. And the phones have a couple of runtimes for the different versions, and then you pick one of the versions you want to develop against – usually the newest version, or maybe an older one. So we kind of want to adopt the same scheme, but make it more pluralistic. In Linux everything is pluralistic.