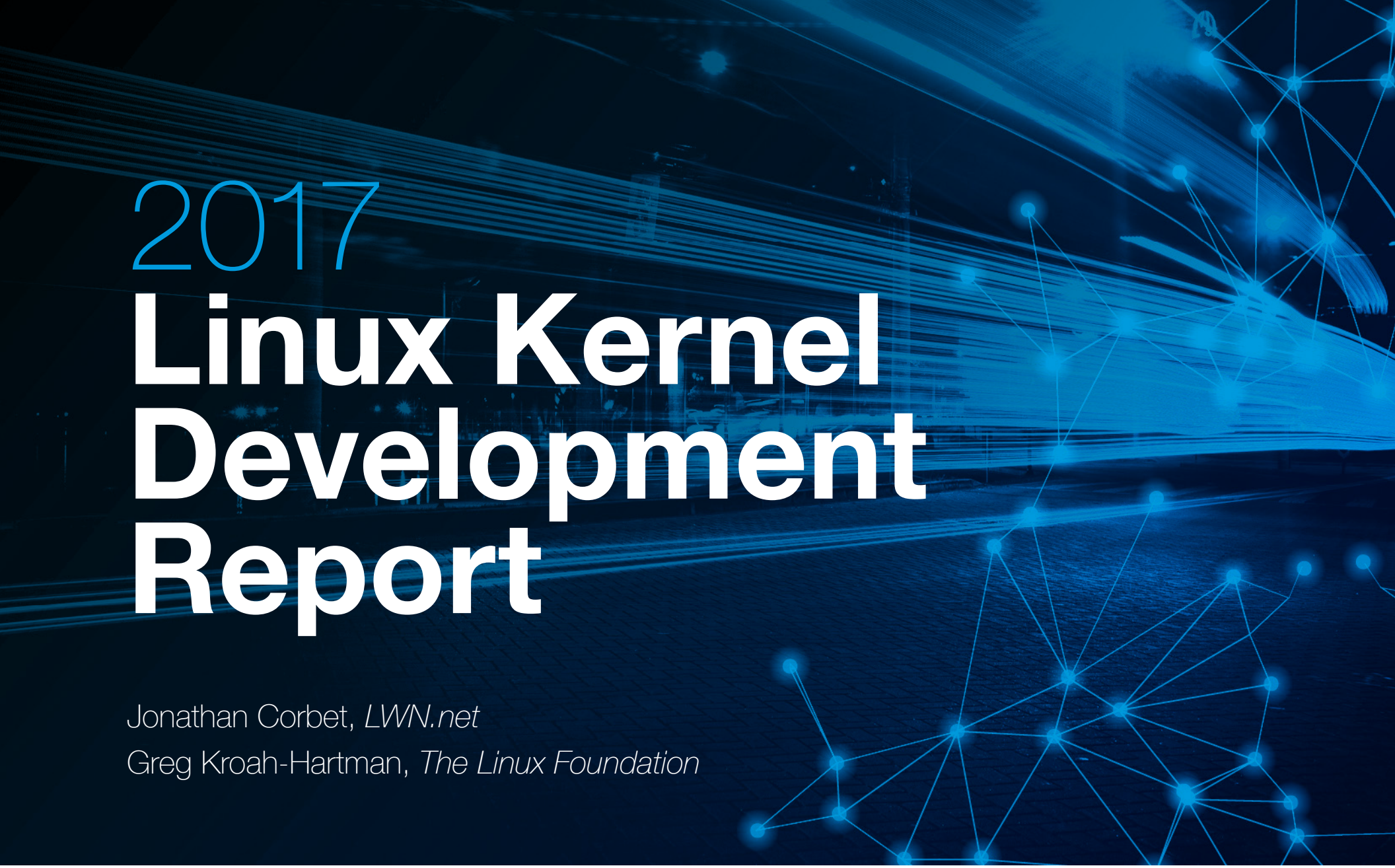


2017 Linux Kernel Development Report

Jonathan Corbet, *LWN.net*
Greg Kroah-Hartman, *The Linux Foundation*

THE LINUX FOUNDATION

Table of Contents

Summary

Linux has come to dominate the open source software world as one of the most successful collaborative development projects in history. As it grows, Linux also comes to dominate nearly every market it enters, including cloud, mobile, embedded, and supercomputing.

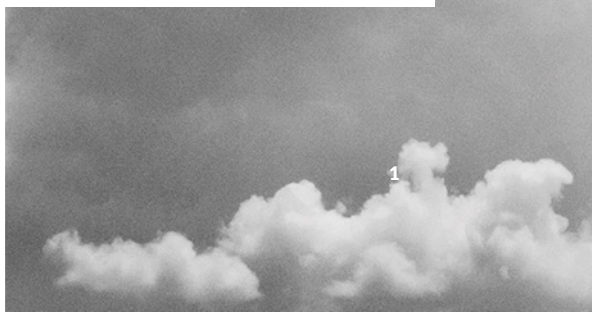
As of 2017, the Linux operating system runs 90 percent of the public cloud workload, has 62 percent of the embedded market share, and 99 percent of the supercomputer market share. It runs 82 percent of the world's smartphones and nine of the top ten public clouds. However, the sustained growth of this open source ecosystem and the amazing success of Linux in general would not be possible without the steady development of the Linux kernel.

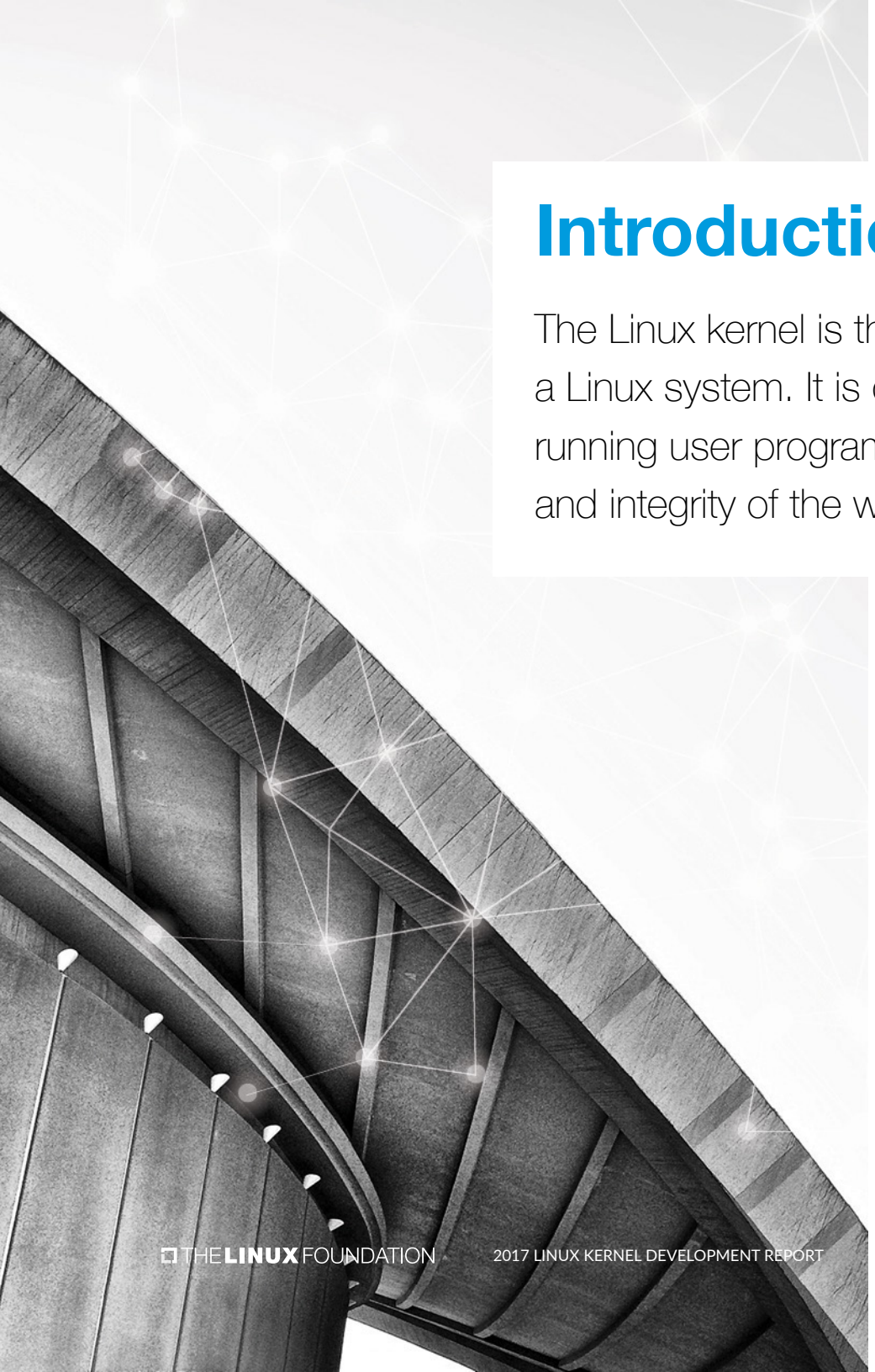
The Linux kernel, which forms the core of the Linux system, is the result of one of the largest cooperative software projects ever attempted. Regular releases every nine to ten weeks deliver stable updates to Linux users, each with significant new features, added device support, and improved performance. The rate of change in the kernel is high and increasing, with over 12,000 patches going into each recent kernel release. Each of these releases contains the work of over 1,600 developers representing over 200 corporations.

Since 2005, some 15,600 individual developers from over 1,400 different companies have contributed to the kernel. The Linux kernel has become a common resource developed on a massive scale by companies which are fierce competitors in other areas.

This is the eighth in a series of regular updates to this document, which has been published roughly annually since 2008. It covers development through the 4.13 release (which came out on September 3, 2017), with an emphasis on the releases (4.8 to 4.13) made since the last update. It has been a typically busy period, with six kernel releases created over a period of about 14 months, many significant changes made, and continual growth of the kernel developer and user community.

The Linux kernel is not a young project; it just celebrated its 26th anniversary. The software world is one of constant change, and today's hot system often doesn't last past tomorrow, but Linux, after more than a quarter of a century, is going stronger than ever. Clearly, the kernel developers are doing something right. This report provides an update into what those developers have been doing and why they continue to be successful.





Introduction

The Linux kernel is the lowest level of software running on a Linux system. It is charged with managing the hardware, running user programs, and maintaining the overall security and integrity of the whole system.

It is this kernel which, after its initial release by Linus Torvalds in 1991, jump-started the development of Linux as a whole. The kernel is a relatively small part of the software on a full Linux system (many other large components come from the GNU project, the GNOME and KDE desktop projects, the X.org project, and many other sources), but it is the core which determines how well the system will work and is the piece which is truly unique to Linux.

The Linux kernel is an interesting project to study for a number of reasons. It is one of the largest individual components on almost any Linux system. It also features one of the fastest-moving development processes and involves more developers than any other open source project. Since 2005, kernel development history is also quite well documented, thanks to the use of the Git source code management system.

Some development highlights since 4.7

The kernel development community remains extremely busy, as will be seen in the statistics shown on the following pages. Some of the highlights from the period since the 4.7 release include:

Changesets merged

Just under 83,000 changesets have been merged from 4,319 individual developers representing 519 corporations (that we know about). The number of changesets per release (in other words, the rate of change of the kernel) and number of developers have both increased from the previous report; the number of companies involved remains steady.

New features merged

As usual, a wide array of new features has been merged during this time period, including:

- The transparent huge page feature now works with file-backed pages as well as program-data pages, leading to much more efficient use of memory.
- The kernel's documentation system was replaced with a new toolchain backed by **Sphinx**; work continues to better organize and enhance our documentation.
- The kernel's core timer mechanism was replaced with a far more efficient implementation.
- The "express data path" mechanism in the networking stack enables high-speed packet processing with user-loaded BPF programs.
- The BBR congestion-control algorithm will improve networking performance in a number of settings.
- Support for Intel's cache-allocation technology gives better control over performance for both enterprise and realtime workloads.
- The swapping subsystem has greatly improved scalability — important now that persistent-memory devices can be used for swapping.
- Support for persistent memory, which may fundamentally change what we can do with our computers, has been greatly improved in general.
- The long-awaited statx() system call is now available.
- The BFQ and Kyber block I/O schedulers provide better performance for a wide variety of I/O workloads.
- The new TEE framework facilitates the use of trusted execution environments on ARM processors.
- The in-kernel TLS implementation allows the offloading of encrypted network streams.

New drivers

Naturally, the kernel developers also added hundreds of new drivers and thousands of fixes over the past year.

Hardening

Work on hardening the kernel against attack continues with the addition of several new technologies, many of which have their origin in the grsecurity and PaX patch sets. New hardening features include virtually mapped kernel stacks, the use of the GCC plugin mechanism for structure-layout randomization, the hardened usercopy mechanism, and a new reference-count mechanism that detects and defuses reference-count overflows. Each of these features makes the kernel more resistant to attack.

Testing

The kernel testing infrastructure continues to improve. The "zero-day build and boot robot" system alone found 223 bugs (all of which were fixed) during this period. The in-kernel self-test framework continues to develop and will someday be a comprehensive test suite for the kernel.

Record numbers

The 4.9 development cycle was the busiest in the kernel's history with regard to the number of changes merged. 4.12, instead, set a new record for both the number of developers involved and the number of first-time contributors. After all these years, the kernel development community is still growing.

Above and beyond all of that, though, the process of developing the kernel and making it better continued at a fast pace. The remainder of this report will concern itself with the health of the development process and where all that code came from.

"I enjoy the work. It's challenging and fun, plus there's a personal gratification knowing that your code is running on billions of devices."

- Jens Axboe, Software engineer, *Facebook*

Development model

Linux kernel development proceeds under a loose, time-based release model, with a new major kernel release occurring every nine or ten weeks.

This model, which was first formalized in 2005, gets new features into the mainline kernel and out to users with a minimum of delay. That, in turn, speeds the pace of development and minimizes the number of external changes that distributors need to apply. As a result, most distributor kernels contain relatively few distribution-specific changes; this leads to higher quality and fewer differences between distributions.

After each mainline release, the kernel's "stable team" (currently led by Greg Kroah-Hartman) takes up short-term maintenance, applying important fixes as they are developed. The stable process ensures that important fixes are made available to distributors and users and that they are incorporated into future mainline releases as well. In recent years, we have seen an increasing number of cooperative industry efforts to maintain specific kernels for periods of one year or more.

Release frequency

The desired release period for a major kernel release is, by common consensus, eight to twelve weeks. A much shorter period would not give testers enough times to find problems with new kernels, while a longer period would allow too much work to pile up between releases. Over the years, the length of the development cycle has stabilized at nine or ten weeks, making kernel releases quite predictable. This can be seen in the release history since 4.7, at right.

Kernel Version	Release Date	Days of Development
4.8	2016-10-02	70
4.9	2016-12-11	70
4.10	2017-02-19	70
4.11	2017-04-30	70
4.12	2017-07-02	63
4.13	2017-09-03	63

Whether a development cycle requires nine or ten weeks is partially dependent on how quickly the kernel stabilizes; sometimes an extra week is required tochase down some final issues. In other cases, as happened with 4.10, the extra week is added to avoid opening a new merge window during an inconvenient time (when key developers are traveling, for example).

The trend toward shorter, more predictable release cycles is almost certainly the result of improved discipline both before and during the development cycle: higher-quality patches are being merged, and the community is doing a better job of fixing regressions quickly. The increased use of automatic testing tools is also helping the community to find (and address) problems more quickly.

Rate of Change

When preparing work for submission to the Linux kernel, developers break their changes down into small, individual units, called "patches." These patches usually do only one thing to the source code; they are built on top of each other, modifying the source code by changing, adding, or removing lines of code. Each patch should, when applied, yield a kernel which still builds and works properly. This discipline forces kernel developers

Kernel Version	Changes (patches)
4.8	13,382
4.9	16,214
4.10	13,029
4.11	12,724
4.12	14,570
4.13	13,006

to break their changes down into small, logical pieces; as a result, each change can be reviewed for code quality and correctness. One other result is that the number of individual changes that go into each kernel release is large and increasing, as can be seen in the table at left.

It was an exceedingly busy year for the kernel development community, with almost 83,000 patches merged overall. The 4.9 development cycle was the busiest in the kernel project's history, partly as a result of the merging of the greybus infrastructure. 4.12, meanwhile, is the second-busiest cycle ever.

By taking into account the amount of time required for each kernel release, one can arrive at the number of changes accepted into the kernel per hour. The results can be seen in at far right.

Over the entire 406-day period covered by this report, the community merged changes at an average rate of 8.5 patches/hour — a significant increase from the 7.8 patches per hour noted in the previous report. The 4.9 and 4.12 development cycles featured the highest patch rates ever seen in the history of the kernel project.

It is worth noting that the above figures understate the total level of activity; most patches go through a number of revisions before being accepted into the mainline kernel, and many are never accepted at all. The ability to sustain this rate of change for years is unprecedented in any previous public software project.

Stable updates

Kernel development does not stop with a mainline release. Inevitably, problems will be found in released kernels, and patches will be made to fix those problems. The stable kernel update process was designed to capture those patches in a way that ensures that both the mainline kernel and current releases are fixed. These stable updates are the base from which most distributor kernels are made.

Kernel Version	Changes per hour
4.8	7.97
4.9	9.65
4.10	7.63
4.11	7.57
4.12	9.64
4.13	8.52

The recent stable kernel update history looks like this:

Kernel	Updates	Fixes
3.18	76	5,264
4.4	93	5,579
4.8	17	1,086
4.9	57	4,035
4.10	17	1,122
4.11	12	982
4.12	14	827
4.13	8	556

The normal policy for stable releases is that each kernel will receive stable updates for a minimum of one development cycle (actually, until the -rc1 release of the second cycle following the initial release). About once each year, one release is chosen to receive updates for an extended, two-year period; as of this writing, the and 4.4 and 4.9 kernels are being maintained in this manner. The 3.18 kernel is used in many mobile devices and is receiving extended maintenance for now. The next long-term support kernel is expected to be 4.14.

It is worth noting that several other kernel releases have been adopted for stable maintenance outside of the normal stable process. The purpose and scope of these long-term kernels varies. The oldest of these long-term releases is currently 3.2, maintained by Debian kernel developer Ben Hutchings; it has had 94 releases incorporating 8,092 fixes.

One might wonder why Linux kernel releases require hundreds or even thousands of fixes after they are declared "finished." There are many problems that are tied to specific hardware or workloads that the developers have no access to. So there will always be issues that come up once a kernel is made available and receives wider testing. Despite the progress that has been made in kernel testing, the community will always be dependent on its users to run tests and report problems.

In the end, most Linux users are running a kernel based off one of the stable updates; to do otherwise would be to miss out on large numbers of important fixes. The stable update series continues to prove its value by allowing the final fixes to be made to released kernels while, simultaneously, letting mainline development move forward.

Laura Abbott

Fedora Kernel Engineer, *Red Hat*

What's your role?

My full time job is working as one of two maintainers for the Fedora kernels. This means I push out kernel releases and fix/shepherd bugs. Outside of that role, I maintain the Ion memory management framework and do occasional work on arm/arm64 and KSPP (kernel hardening)

What does the kernel community need to work on?

As a general theme, there needs to be a focus on scaling the community. There's always an ongoing discussion about how to attract new developers and there's been a recent focus on how to grow contributors into maintainers. I'd like to see the kernel community continue to make processes easier for new and existing developers. I'd also like to see the discussions about building an inclusive community continue.

Why do you contribute?

I've always found low level systems fascinating and enjoy seeing how all the pieces work together. There's always something new to learn about in the kernel and I find the work challenging.

Jens Axboe

Software engineer, *Facebook*

What's your role?

I'm the Linux block layer maintainer, so I primarily develop features in that area, as well as help review and guide others doing the same.

What does the kernel community need to work on?

Attracting more young talent. Most young folks these days gravitate towards product instead of infrastructure. It's important that we bring new talent into the fold.

Why do you contribute?

First of all, because I enjoy the work. It's challenging and fun, plus there's a personal gratification knowing that your code is running on billions of devices. Finally, it's my job.

Kernel source size

The Linux kernel keeps growing in size over time as more hardware is supported and new features are added.

For the following numbers, we have counted everything in the released Linux source package as "source code" even though a small percentage of the total is the scripts used to configure and build the kernel, as well as a fair amount of documentation. Those files, too, are part of the larger work, and thus merit being counted.

Version	Files	Lines
4.8	55,472	22,070,760
4.9	56,201	22,348,062
4.10	57,167	22,839,361
4.11	57,959	23,137,101
4.12	59,801	24,170,555
4.13	60,538	24,766,703

The information in the table above shows the number of files and lines in each kernel version.

The kernel has grown steadily since its first release in 1991, when there were only about 10,000 lines of code. At almost 25 million lines (up from nearly 22 million), the kernel is almost three million lines larger than it was at the time of the previous version of this report. Another way of putting this is that, in the production of the 4.8 to 4.13 releases, the kernel community added just over 15 files and nearly 7,500 lines of code — every day.

Arnd Bergmann

Linaro

What's your role?

I co-maintain the arm-soc kernel tree together with Olof Johansson. This is where all the platform specific changes for ARM processors and SoCs (both 32-bit and 64-bit) get merged. This is one of the larger subsystems in the kernel, and it interacts with most driver subsystems.

I also maintain a couple of smaller things in the kernel. In particular, I look after new CPU architectures getting merged into the kernel and the associated include/asm-generic/ directory.

One of my long-term projects is to fix the time_t overflow in the kernel, which will cause 32-bit code to fail in the year 2038.

What have you been working on recently?

Aside from my maintainer work, I have spent a lot of time during the last year on fixing hundreds of smaller bugs that lead to build failures or warnings. I started doing a lot of build-testing as a way to improve the quality of contributions I merge into the kernel from other people, but this has now turned into a separate effort to get all random configurations to build cleanly.

Mauro Carvalho Chehab

Open Source Director, Samsung Research Brazil

What's your role?

I'm responsible for the Open Source efforts at Samsung Research Brazil, as part of Samsung's Open Source Group. I maintain the media and EDAC (Error Detection and Correction) Kernel subsystems.

What have you been working on this year?

This year, I did a lot of patches that improve Linux documentation. A lot of them were related to the conversion from the XML-based DocBook docs to a markup language (Restructured Text). Thanks to that, no documents use the legacy document system anymore. I also finally closed the documentation gap at the DVB API, with was out of sync for more than 10 years! I also did several bug fixes at the media subsystem, including the 4.9 breakage of many drivers that were doing DMA via stack.

Why do you contribute?

Because it is fun! Seriously, I strongly believe that the innovation process in computer engineering are currently driven by Linux. Working on its Kernel has provided me the opportunity of working with great developers and helping to improve a cutting edge operating system.

Who is doing the work

The number of different developers who are doing Linux kernel development and the identifiable companies who are sponsoring this work have been increasing over the different kernel versions, as can be seen in the following table.

Version	Developers	Companies
4.8	1,597	262
4.9	1,729	270
4.10	1,680	273
4.11	1,741	268
4.12	1,821	274
4.13	1,681	225

These numbers show a continuation of the steady increase in the number of developers contributing to each kernel release, with an all-time record being set with the 4.12 release.

Since the beginning of the git era (the 2.6.11 release in 2005), a total of 15,637 developers have contributed to the Linux kernel; those developers worked for a minimum of 1,513 companies. The number of companies supporting work on the kernel appears to be stable and not growing like the number of developers — but the 273 companies observed to have supported work on 4.10 was an all-time record.

Despite the large number of individual developers, there is still a relatively small number who are doing the majority of the work. In any given development cycle, approximately one-third of the developers involved contribute exactly one patch. Since the 2.6.11 release, the top 10 developers together have contributed 45,338 changes — almost 7.1 percent of the total. The top 30 developers contributed just under 16 percent of the total. Those developers are listed here.

Top 30 kernel developers, 2.6.11–4.13

Name	Changes	%
H Hartley Sweeten	6,034	0.9%
Al Viro	5,904	0.9%
Takashi Iwai	5,089	0.8%
Mauro Carvalho Chehab	5,039	0.8%
David S. Miller	4,044	0.6%
Johannes Berg	4,014	0.6%
Mark Brown	3,978	0.6%
Tejun Heo	3,951	0.6%
Russell King	3,692	0.6%
Greg Kroah-Hartman	3,593	0.6%
Thomas Gleixner	3,582	0.6%
Christoph Hellwig	3,498	0.5%
Hans Verkuil	3,419	0.5%
Ingo Molnár	3,128	0.5%
Chris Wilson	3,090	0.5%
Arnd Bergmann	3,071	0.5%
Geert Uytterhoeven	3,011	0.5%
Dan Carpenter	2,994	0.5%
Eric Dumazet	2,988	0.5%
Joe Perches	2,937	0.5%
Alex Deucher	2,757	0.4%
Daniel Vetter	2,688	0.4%
Laurent Pinchart	2,687	0.4%
Axel Lin	2,670	0.4%
Trond Myklebust	2,554	0.4%
Ben Skeggs	2,516	0.4%
Arnaldo Carvalho de Melo	2,456	0.4%
Bartlomiej Zolnierkiewicz	2,331	0.4%
Kuninori Morimoto	2,300	0.4%
Linus Walleij	2,281	0.4%

The numbers in the previous table are drawn from the entire git repository history, starting with 2.6.12. If we look at the commits since the last version of this report (4.7) through 4.13, the picture is somewhat different.

Note that many senior kernel developers, Linus Torvalds included, do not show up on these lists. These developers spend much of their time getting other developers' patches into the kernel; this work includes reviewing changes and routing accepted patches toward the mainline. We will look more closely at that work below.

Top 30 kernel developers, 4.8–4.13

Name	Changes	%
Chris Wilson	1,519	1.8%
Mauro Carvalho Chehab	1,096	1.3%
Johan Hovold	911	1.1%
Arnd Bergmann	872	1.1%
Christoph Hellwig	801	1.0%
Geert Uytterhoeven	551	0.7%
Viresh Kumar	550	0.7%
Colin Ian King	539	0.6%
Ville Syrjälä	494	0.6%
Wei Yongjun	493	0.6%
Greg Kroah-Hartman	476	0.6%
Al Viro	471	0.6%
Daniel Vetter	464	0.6%
Dan Carpenter	432	0.5%
Masahiro Yamada	429	0.5%
Kuninori Morimoto	428	0.5%
Markus Elfring	427	0.5%
Alex Deucher	417	0.5%
Linus Walleij	404	0.5%
Javier Martinez Canillas	384	0.5%
Takashi Iwai	366	0.4%
Eric Dumazet	355	0.4%
Wolfram Sang	352	0.4%
Andy Shevchenko	351	0.4%
Thomas Gleixner	349	0.4%
Arnaldo Carvalho de Melo	349	0.4%
Alex Elder	345	0.4%
David Howells	342	0.4%
Hans de Goede	333	0.4%
Florian Fainelli	323	0.4%

"I contribute to the Linux kernel because it is fun. While it is difficult for a single company to employ all the smartest people, I can reach out to the kernel development mailing list to debate and collaborate with world class talent. The debates inevitably make progress because we are always unified by the common principle of doing what is best for the long term health of the project. I grow and learn something new almost every time I make a contribution."

- Dan Williams, Software Engineer, *Intel*

Kees Cook

Software Engineer, *Google*

What's your role?

Recently, I organized the Kernel Self-Protection Project (KSPP), which has helped focus lots of other developers to work together to harden the kernel against attack. I'm also the maintainer of seccomp, pstore, LKDTM, and gcc-plugin subsystems, and a co-maintainer of sysctl.

What does the kernel community need to work on?

I think we've got a lot of work ahead in standardizing the definitions of syscalls and continuing to identify and eliminate error-prone code patterns. Doing these kinds of tree-wide changes continues to be quite a challenge for contributors because the kernel development model tends to focus on per-subsystem development.

Why do you contribute?

I've always loved working with low-level software, close to the hardware boundary. I love the challenges it presents. Additionally, since Linux is used in all corners of the world, it's hard to find a better project to contribute to that has such an impact on so many people's lives.

Thomas Gleixner

CTO, *Linutronix GmbH*

What's your role?

I serve various maintainer roles. The x86 architecture, the generic interrupt subsystem, and the time(r) subsystem.

What does the kernel community need to work on?

Aside of the technical challenges, which are hard to predict, we need more effort on code cleanup and consolidation along with more capacity for reviews.

Why do you contribute?

First of all it's fun, and I strongly believe that FOSS is the right way to go, but I freely admit that I also do it to earn my living.

Who is sponsoring the work

The Linux kernel is a resource which is used by a large variety of companies.

Many of those companies never participate in the development of the kernel; they are content with the software as it is and do not feel the need to help drive its development in any particular direction. But, as can be seen in the table above, an increasing number of companies are working toward the improvement of the kernel.

Below we look more closely at the companies which are employing kernel developers. For each developer, corporate affiliation was obtained through one or more of: the use of company email addresses, sponsorship information included in the code they submit, or simply asking the developers directly. The numbers presented are necessarily approximate; developers occasionally change employers, and they may do personal work out of the office. But they will be close enough to support a number of conclusions.

There are a number of developers for whom we were unable to determine a corporate affiliation; those are grouped under "unknown" in the table at right With few exceptions, all of the people in this category have contributed 10 or fewer changes to the kernel over the past three years, yet the large number of these developers causes their total contribution to be quite high.

The category "none," instead, represents developers who are known to be doing this work on their own, with no financial contribution happening from any company.

The most active companies over the 4.8 to 4.13 development cycles are listed here.

Top companies contributing to the Linux kernel, 4.8–4.13

Company	Changes	%
Intel	10,833	13.1%
none	6,819	8.2%
Red Hat	5,965	7.2%
Linaro	4,636	5.6%
unknown	3,408	4.1%
IBM	3,359	4.1%
consultants	2,743	3.3%
Samsung	2,633	3.2%
SUSE	2,481	3.0%
Google	2,477	3.0%
AMD	2,215	2.7%
Renesas Electronics	1,680	2.0%
Mellanox	1,649	2.0%
Oracle	1,402	1.7%
Huawei Technologies	1,275	1.5%
Broadcom	1,267	1.5%
ARM	1,256	1.5%
Texas Instruments	1,136	1.4%
Free Electrons	969	1.2%
NXP Semiconductors	839	1.0%
Canonical	805	1.0%
Facebook	771	0.9%
Imagination Technologies	669	0.8%
Cavium	664	0.8%
Code Aurora Forum	648	0.8%
Outreachy	633	0.8%
BayLibre	615	0.7%
NVidia	579	0.7%
linutronix	565	0.7%
Rockchip	507	0.6%

"I've always found low level systems fascinating and enjoy seeing how all the pieces work together. There's always something new to learn about in the kernel and I find the work challenging."

- Laura Abbott, Fedora Kernel Engineer, *Red Hat*

The top 10 contributors, including the groups "unknown" and "none," make up just over 54 percent of the total contributions to the kernel; that is up slightly from the previous version of this report. It is worth noting that, even if one assumes that all of the "unknown" contributors are working on their own time, well over 85 percent of all kernel development is demonstrably done by developers who are being paid for their work.

Interestingly, the volume of contributions from unpaid developers has been in slow decline for many years. It was 14.6 percent in the 2012 version of this report, but is 8.2 percent this time around. There are many possible reasons for this decline, but, arguably, the most plausible of those is quite simple: kernel developers are in short supply, so anybody who demonstrates an ability to get code into the mainline tends not to have trouble finding job offers. Indeed, the bigger problem can be fending those offers off. As a result, volunteer developers tend not to stay that way for long.

What we see here is that a small number of companies is responsible for a large portion of the total changes to the kernel. But there is a "long tail" of companies (nearly 500 of which do not appear in the above list) which have made significant changes since the 4.7 release. There may be no other examples of such a large, common resource being supported by such a large group of independent actors in such a collaborative way.

Shuah Khan

Sr. Linux Kernel Developer, *Samsung Research America*

What's your role?

I'm a contributor, maintainer, and I serve on the Linux Foundation Technical Advisory Board. I maintain the Linux Kernel Selftest framework and USB-over-IP driver. I also contribute to the Linux Media, Power Management, IOMMU, and DMA areas.

What have you been working on recently?

My main focus this year has been Exynos platform upstream stability, Kselftest framework and individual tests. I contributed to improving the quality of media subsystem core, and media and drm drivers on Exynos platform. I enhanced and improved the Kselftest framework by adding support for the Test Anything Protocol and object relocation. And I boot tested stable kernel release candidates and maintained the Kselftest and USB-over-IP drivers.

What does the kernel community need to work on?

The Linux Kernel community should continue its focus on adding support for new hardware, harden the security, and improve quality. Focusing on effective ways to proactively detect security vulnerabilities, race conditions, and hard to find problems will help achieve the above goals.

Julia Lawall

Senior Researcher, *Inria*

What's your role?

I work on the tool Coccinelle that is used to find bugs in the Linux kernel and perform large-scale evolutions. I also coordinate the Linux kernel projects for the Outreachy internship program.

What have you been working on recently?

This year I have been working with Bhumika Goyal on making various kernel structures read-only, supported in part by the Core Infrastructure Initiative (CII). I have also been working on automatically identifying patches that should be considered for backporting to stable kernels, in collaboration with Greg KH, Sasha Levin, and colleagues at Singapore Management University. Our approach is still work in progress, but several hundred commits that were not originally tagged for stable have been identified and applied to stable versions.

Why do you contribute?

Many reasons: the potential impact, the challenge of understanding a huge code base of low-level code, the chance to interact with a community with a very high level of technical skill.

Bringing in new developers

The decline in volunteer developers mentioned in the previous section is potentially a cause for concern.

Many, if not most, of the current development community started that way, after all; might a shortage of volunteers lead to a shortage of kernel developers in the future? The situation is worth watching, but there are a number of reasons to not worry too much about it at this time. The first of those was mentioned above: successful volunteers tend not to stay volunteers for long; why do the work for free when somebody is willing to pay for it? But there is more to the story than that.

Release	New Developers
4.8	235
4.9	300
4.10	261
4.11	280
4.12	331
4.13	263

Over the course of kernel development since the use of Git began, each kernel release has included contributions from 200-300 developers who had never put a patch into the kernel before. Outliers include 2.6.25 (333 new developers) and 2.6.20 (169 new developers). For the time period covered by this report, the history is at left.

That adds up to 1,670 first-time developers over the course of just under 13 months. Remember that 4,319 developers overall contributed to the kernel during this time; one can thus conclude that over one-third of them were contributing for the first time. Many of those developers will get their particular fix merged and never be seen again, but others will become permanent members of the kernel development community.

Of those 1,670 new developers, 266 were known to be working on their own time, while we have not yet been able to get information on 405 of them. The rest of the new developers (999, representing 60 percent of the total) were already working for a company when they contributed their first patch to the kernel. The companies that have been most active in bringing new developers into the community are shown at right.

Company	New Developers
Intel	128
Google	58
Huawei Technologies	33
Code Aurora Forum	33
IBM	31
Mellanox	28
AMD	26
Samsung	24
NXP Semiconductors	21
Cavium	20

The bottom line is that, even if all of the unknowns were volunteers, more than half of our new developers are paid to work on the kernel from their very first patch. In other words, companies working in this area have realized that one of the best ways to find new kernel development talent is to develop it in-house. So, for many developers, employment comes first, and it is no longer necessary to put in time as a volunteer developer. This fact, too, can explain the decrease in volunteers over time while simultaneously showing that the community as a whole remains healthy.

The most popular area for new developers to make their first patch is the "staging tree," a place for device drivers that have not yet reached sufficient quality to merit inclusion in the kernel proper. One of the reasons for the creation of staging was to make life easy for beginning developers who want to make simple improvements to the code; this result suggests that it has been successful in that goal.

Beyond staging, the most popular parts of the kernel for new developers include networking (both the networking core and networking drivers), documentation (which has seen a significant increase recently, suggesting that recent efforts to make kernel documentation more accessible are having an effect), graphics drivers, USB drivers, and the audio subsystem.

"I contribute to the kernel for many reasons: the potential impact, the challenge of understanding a huge code base of low-level code, the chance to interact with a community with a very high level of technical skill."

- Julia Lawall, Senior Researcher, *Inria*

"I strongly believe that the innovation process in computer engineering is currently driven by Linux. Working on its kernel has provided me the opportunity of working with great developers and helping to improve a cutting edge operating system."

- Mauro Carvalho Chehab, Open Source Director, *Samsung Research Brazil*

David Miller

Consulting Engineer, *Red Hat Inc.*

What's your role?

I am the top-level maintainer of the networking subsystem and the Sparc port.

What have you been working on recently?

I removed UFO from the networking code.

Why do you contribute?

I'm locked in a room, and not allowed to leave. Sometimes they push some food underneath the door.

Andrew Morton

Member of Technical Staff - Engineering, *Google*

What's your role?

Mainly maintenance of the memory management subsystem. I also act as a maintainer of last resort for subsystems which don't have a current maintainer.

What does the kernel community need to work on?

I think we're doing OK, so largely steady-as-she-goes. I continue to worry that the kernel is becoming more and more complex and hence less and less approachable for new developers. I wish we'd be more careful about decreasing the technical barriers as we go about our day-to-day development.

Why do you contribute?

It's what I do, and I like to think that my little efforts are helping people and organizations.

Steven Rostedt

Open Source Programmer, *VMware*

What's your role?

I'm an Open Source advocate and try to communicate to people what that means. I maintain the Real Time Stable releases, and the Ftrace (Linux kernel tracer) subsystem. I'm on The Linux Foundation's Technical Advisory Board.

What does the kernel community need to work on?

I think more focus should be on eBPF and helping it be easier to use as well as having an eye on security. Running a VM within the kernel can be very dangerous, and people need to use caution and be extra careful during development.

Why do you contribute?

Because it is the one place that you have total control over your computer.

Who is reviewing the work

Patches do not normally pass directly into the mainline kernel; instead, they pass through one of over 100 subsystem trees.

Each subsystem tree is dedicated to a specific part of the kernel (examples might be SCSI drivers, x86 architecture code, or networking) and is under the control of a specific maintainer. When a subsystem maintainer accepts a patch into a subsystem tree, he or she will attach a "Signed-off-by" line to it. This line is a statement that the patch can be legally incorporated into the kernel; the sequence of signoff lines can be used to establish the path by which each change got into the kernel.

An interesting (if approximate) view of kernel development can be had by looking at signoff lines, and, in particular, at signoff lines added by developers who are not the original authors of the patches in question. These additional signoffs are usually an indication of review by a subsystem maintainer. Analysis of signoff lines gives a picture of who admits code into the kernel — who the gatekeepers are.

Since 4.7, the developers who added the most non-author signoff lines are at right.

The total number of patches signed off by Linus Torvalds (207, or 0.3 percent of the total) continues its long-term decline. That reflects the increasing amount of delegation to subsystem maintainers who do the bulk of the patch review and merging.

Associating signoffs with employers yields the following:

Developers with the most non-author signoffs, since 4.7

Developer	Signoffs	%
David S. Miller	9,032	11.6%
Greg Kroah-Hartman	8,416	10.8%
Mark Brown	2,289	2.9%
Andrew Morton	2,099	2.7%
Mauro Carvalho Chehab	1,997	2.6%
Alex Deucher	1,732	2.2%
Ingo Molnár	1,718	2.2%
Martin K. Petersen	1,482	1.9%
Kalle Valo	1,294	1.7%
Jens Axboe	1,226	1.6%
Doug Ledford	1,216	1.6%
Michael Ellerman	1,184	1.5%
Linus Walleij	979	1.3%
Jonathan Cameron	901	1.2%
Arnaldo Carvalho de Melo	889	1.1%
Rafael J. Wysocki	860	1.1%
Herbert Xu	847	1.1%
Thomas Gleixner	786	1.0%
Daniel Vetter	766	1.0%
Ulf Hansson	715	0.9%

Companies associated with the most signoffs, since 4.7

Company	Signoffs	%
Red Hat	16,132	20.6%
Intel	7,589	9.7%
Linux Foundation	7,110	9.1%
Linaro	6,158	7.9%
Google	5,750	7.4%
none	3,073	3.9%
Samsung	2,938	3.8%
IBM	2,391	3.1%
SUSE	1,879	2.4%
AMD	1,838	2.4%
Oracle	1,791	2.3%
Facebook	1,563	2.0%
Code Aurora Forum	1,532	2.0%
Mellanox	1,375	1.8%
Free Electrons	1,223	1.6%
ARM	991	1.3%
unknown	916	1.2%
linutronix	867	1.1%
Renesas Electronics	786	1.0%
Cisco	736	0.9%

The signoff metric is a loose indication of review, so the above numbers need to be regarded as approximations only. Still, one can clearly see that subsystem maintainers are rather more concentrated than kernel developers as a whole; over half of the patches going into the kernel pass through the hands of developers employed by just five companies. Over the years, the community of subsystem maintainers has become less concentrated, but it is a slow process.

"I have spent a lot of time during the last year on fixing hundreds of smaller bugs that lead to build failures or warnings. I started doing a lot of build-testing as a way to improve the quality of contributions I merge into the kernel from other people, but this has now turned into a separate effort to get all random configurations to build cleanly."

- Arnd Bergmann, *Linaro*

Daniel Vetter

Staff Engineer, *Intel*

What's your role?

Graphics. Former maintainer.

What have you been working on recently?

Lots of documentation updates in graphics. Mentoring new committers and maintainers. Improving process and community. Some technical stuff to relax.

Why do you contribute?

It pays well. Within graphics it's actually fun; outside, the dread often weighs much heavier.

Linus Walleij

Senior Engineer, *Linaro Limited*

What's your role?

I am a subsystem maintainer for the pin control and GPIO subsystems in the Linux kernel. I also work on assorted ARM devices and sometimes on the MTP (Media Transfer Protocol) initiator library for userspace, libmtp.

What does the kernel community need to work on?

Moving all legacy block devices to the multiqueue block infrastructure and delete the old block layer path. Continue to consolidate old architectures like ARM32 and m68k.

Why do you contribute?

Contributing to the kernel, quite obviously, is ultimately fueled by the satisfaction of the human consciousness of being recognized for your talent by equal peers. This is the social psychological mechanism that drives all intellectual work.

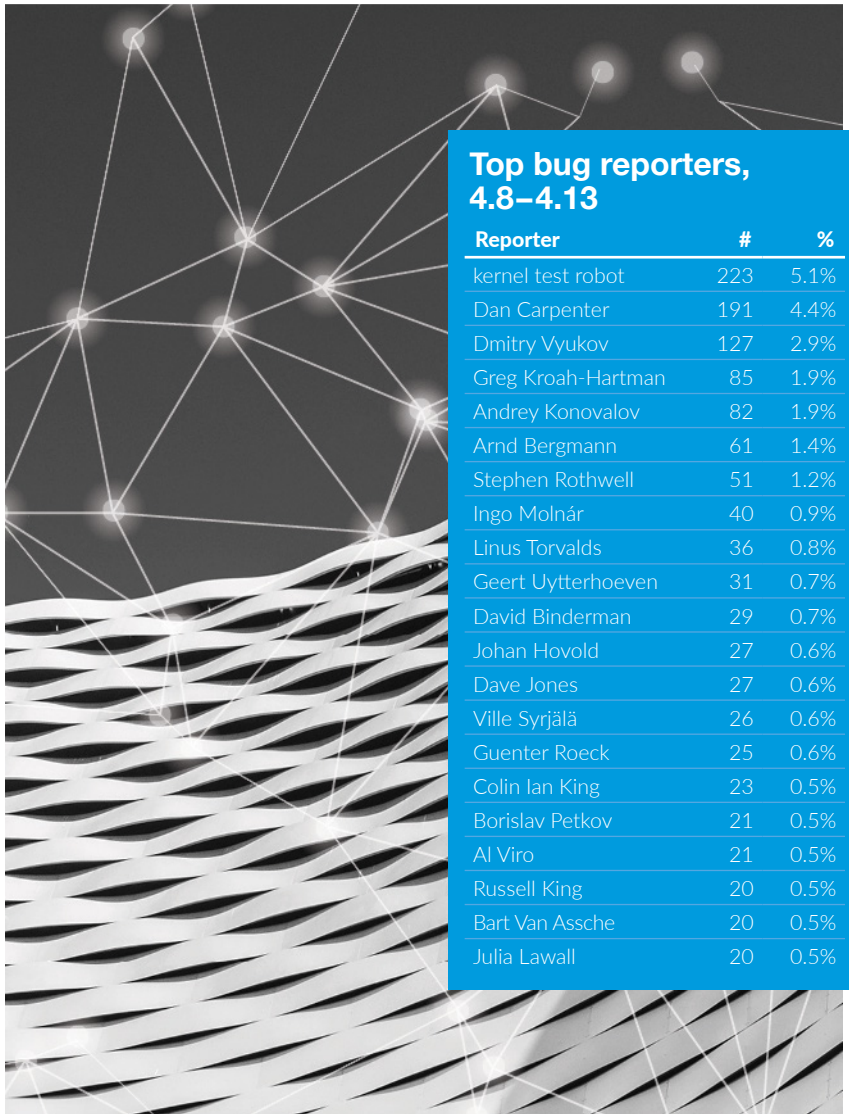
Bug reporting

A kernel that has had nearly 83,000 patches applied to it will certainly have a few bugs introduced along with the new features.

The community depends heavily on a wide community of testers to find those bugs and report them so that they can be fixed. Kernel conventions say that, when a patch fixing a bug is applied to the kernel, it should contain a "Reported-by" tag crediting the tester who found the problem. During the period covered by this report, just over 4,100 patches carried such tags. From those tags, we can determine that the most prolific bug reporters were those at right:

It turns out that the person providing the most bug reports is not a person at all — it's the **0-Day test service** run by Intel. This service picks up patches from the mailing lists and tests them, often before they are accepted for inclusion; as a result, many problems are headed off before they can affect users. Linus Torvalds appears on this list because he routinely boots the kernel that results after accepting a pull request. There are few things that move faster than a kernel developer whose patch has just broken Linus's work computer.

The list, at right, is certainly incomplete, in that adding the "Reported-by" tag is an easy step to overlook. Still, it provides a partial picture of the community of testers that the kernel project relies on to report problems.



Top bug reporters, 4.8–4.13

Reporter	#	%
kernel test robot	223	5.1%
Dan Carpenter	191	4.4%
Dmitry Vyukov	127	2.9%
Greg Kroah-Hartman	85	1.9%
Andrey Konovalov	82	1.9%
Arnd Bergmann	61	1.4%
Stephen Rothwell	51	1.2%
Ingo Molnár	40	0.9%
Linus Torvalds	36	0.8%
Geert Uytterhoeven	31	0.7%
David Binderman	29	0.7%
Johan Hovold	27	0.6%
Dave Jones	27	0.6%
Ville Syrjälä	26	0.6%
Guenter Roeck	25	0.6%
Colin Ian King	23	0.5%
Borislav Petkov	21	0.5%
Al Viro	21	0.5%
Russell King	20	0.5%
Bart Van Assche	20	0.5%
Julia Lawall	20	0.5%

Dan Williams

Software Engineer, *Intel*

What's your role?

I am one of the maintainers of the "libnvdimm" sub-system that supports persistent memory (PMEM) hardware devices. I also contribute to the development of DAX and other kernel technologies that enable PMEM.

What does the kernel community need to work on?

Accelerate the growth of a unit test culture for Linux kernel development. There will always be code paths that need hardware and custom environments, but the libnvdimm unit tests are hopefully an example of how drivers can be tested absent hardware.

Why do you contribute?

I contribute to the Linux kernel because it is fun. While it is difficult for a single company to employ all the smartest people, I can reach out to the kernel development mailing list to debate and collaborate with world class talent. The debates inevitably make progress because we are always unified by the common principle of doing what is best for the long term health of the project. I grow and learn something new almost every time I make a contribution.

Rafael Wysocki

Software Engineer, *Intel*

What's your role?

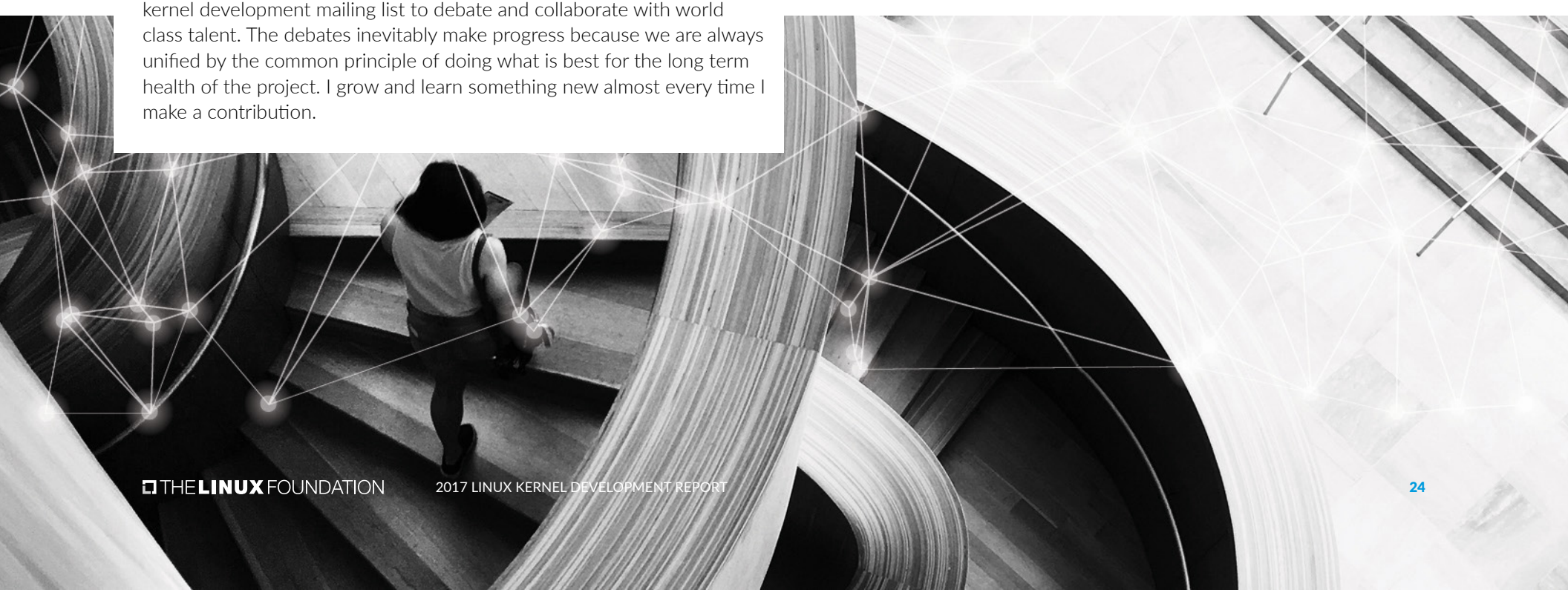
I am the maintainer of the power management infrastructure and the ACPI core in the Linux kernel.

What have you been working on recently?

Power management mostly: cpufreq improvements (schedutil governor and intel_pstate driver improvements) and suspend-to-idle support.

Why do you contribute?

Because I enjoy doing it, I am paid for doing it, and I think that this project is important for the future of civilization as a whole.



Lessons from 26 years of Linux

As was mentioned back at the beginning, the kernel project is by no means young; it celebrated its 25th anniversary in 2016.

Few development projects have that kind of history, and many of those that do have long since settled into a "nearly complete" state where changes are few and far between. The kernel is different, though; after 26 years, this project is more vital and active than at any point in its history. There have been many academic studies of this development community, but it still may be many years before we fully understand the keys to its success. That said, there are a few lessons that stand out clearly:

- **Short release cycles are important.**
  In the early days of the Linux project, a new major kernel release only came once every few years. That meant considerable delays in getting new features to users, which was frustrating to users and distributors alike. But, more importantly, such long cycles meant that huge amounts of code had to be integrated at once, and that there was a great deal of pressure to get code into the next release, even if it wasn't ready.

  Short cycles address all of these problems. New code is quickly made available in a stable release. Integrating new code on a nearly constant basis makes it possible to bring in even fundamental changes with minimal disruption. And developers know that if they miss one release cycle, there will be another one in two months, so there is little incentive to try to merge code prematurely.

- **Process scalability requires a distributed, hierarchical development model.**
  Once upon a time, all changes went directly to Linus Torvalds, but even a developer with his talents cannot keep up with a project moving as quickly as the kernel. Spreading out the responsibility for code review and integration across 100 or more maintainers gives the project the resources to cope with tens of thousands of changes without sacrificing review or quality.

- **Tools matter.**
  Kernel development struggled to scale until the advent of the BitKeeper source-code management system changed the community's practices nearly overnight; the switch to Git brought about another leap forward. Without the right tools, a project like the kernel would simply be unable to function without collapsing under its own weight.

- **The kernel's strongly consensus-oriented model is important.**
  As a general rule, a proposed change will not be merged if a respected developer is opposed to it. This can be intensely frustrating to developers who find code they have put months into blocked on the mailing list, but it also ensures that the kernel remains suited to a wide ranges of users and problems. No particular user community is able to make changes at the expense of other groups. As a result, we have a kernel that scales from tiny systems to supercomputers and that is suitable for a huge range of uses.

- **A related factor is the kernel's strong "no regressions" rule.**
  If a given kernel works in a specific setting, all subsequent kernels must work there, too. The implementation of this rule is not always perfect, but it still gives users assurance that upgrades will not break their systems; as a result, they are willing to follow the kernel as it develops new capabilities.

- **Corporate participation in the process is crucial.**
  We would not have the fast-moving project described here without it. But it is also important that no single company dominates kernel development. While any company can improve the kernel for its specific needs, no company can drive development in directions that hurt the others or restrict what the kernel can do.

- **There should be no internal boundaries within the project.**
  Kernel developers are necessarily focused on specific parts of the kernel, but any developer can make a change to any part of the kernel if the change can be justified. As a result, problems are fixed where they originate rather than being worked around, developers have a wider view of the kernel as a whole, and even the most recalcitrant maintainer cannot indefinitely stall needed progress in any given subsystem.

Above all, 26 years of kernel history show that sustained, cooperative effort can bring about common resources that no group would have been able to develop on its own.

Peter Zijlstra

Software Engineer, *Intel*

What's your role?

I'm (co-)maintainer of the scheduler, the locking infrastructure, the atomic op infrastructure, the performance events (kernel) bits, and I think also the memory model and memory barrier syscall bits, although that's not in MAINTAINERS yet.

What does the kernel community need to work on?

In general, people should work on what they're passionate about. If I were to have any say in the matter at all, I would ask them to take a little more time and consider things from a step back.

Why do you contribute?

Because it's awesome. ;-) That is, you get to meet some very smart people and get to work together on solving some really fun puzzles. And the best part is, everybody can join or learn from it; it's not locked away in some company vault.

Mimi Zohar

Senior Software Engineer, *IBM*

What's your role?

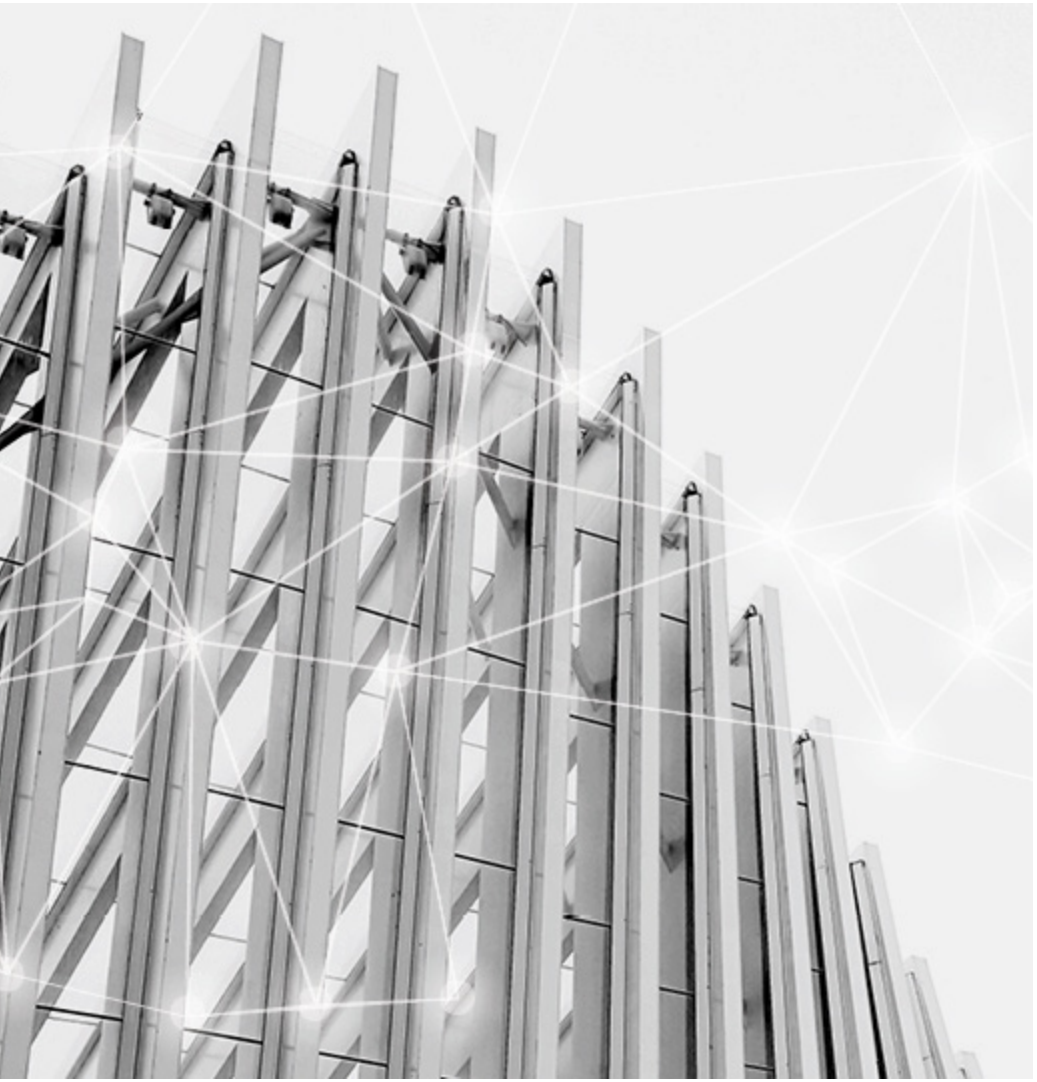
I'm the linux-integrity subsystem maintainer. I work on extending secure and trusted boot to the running OS. There are a number of dependencies on other subsystems (e.g., TPM, trusted and encrypted keys, trusted keyrings).

What have you been working on recently?

We're working on closing measurement/appraisal gaps. For example, files read by the kernel should be measured/appraised. This year, we've extended kexec on openPower to carry the IMA measurement list across kexec to the target OS. This provides a full measurement chain, from boot up to and including the running target OS.

Why do you contribute?

I don't normally hear about who is using the linux-integrity subsystem or how they're using it, only when there are problems. It always surprises me how and where it is being used.



Conclusion

The Linux kernel is one of the largest and most successful open source projects that has ever come about. The huge rate of change and number of individual contributors show that it has a vibrant and active community, constantly causing the evolution of the kernel in response to number of different environments it is used in. This rate of change continues to increase, as does the number of developers and companies involved in the process; thus far, the development process has proved that it is able to scale up to higher speeds without trouble.

There are enough companies participating to fund the bulk of the development effort, even if many companies which could benefit from contributing to Linux have, thus far, chosen not to. With the current expansion of Linux in the server, desktop, mobile and embedded markets, it's reasonable to expect this number of contributing companies – and individual developers – will continue to increase. The kernel development community welcomes new developers; individuals or corporations interested in contributing to the Linux kernel are encouraged to consult **How to Participate in the Linux Community** or to contact the authors of this paper or The Linux Foundation for more information.

Thanks

The authors would like to thank the thousands of individual kernel contributors, without them, papers like this would not be interesting to anyone.

Resources

Many of the statistics in this article were generated by the "gitdm" tool, written by Jonathan Corbet. Gitdm is distributable under the GNU GPL; it can be obtained from **git://git.lwn.net/gitdm.git**.

The information for this paper was retrieved directly from the Linux kernel releases as found at the **kernel.org** web site and from the git kernel repository.