Linux Kernel Development

How Fast It is Going, Who is Doing It, What They Are Doing and Who is Sponsoring the Work

25th Anniversary Edition

A Publication of The Linux Foundation
August 2016

AUTHORS

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

www.linuxfoundation.org

Summary

The kernel which forms the core of the Linux system is the result of one of the largest cooperative software projects ever attempted.

Regular two- to three-month releases 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 10,000 patches going into each recent kernel release. Each of these releases contains the work of more than 1,600 developers representing over 200 corporations.

Since 2005, some 14,000 individual developers from over 1,300 different companies have contributed to the kernel. The Linux kernel, thus, has become a common resource developed on a massive scale by companies which are fierce competitors in other areas.

This document is the seventh in a series of regular updates, which have been published roughly annually since 2008. It covers development through the 4.7 release (which came out on July 24, 2016), with an emphasis on the releases (3.19 to 4.7) made since the last update. It has been a typically busy period, with nine kernel releases created, many significant changes made, and continual growth of the kernel developer and user communities.

The Linux kernel is not a young project; we celebrate its 25th anniversary in 2016. The software world is one of constant change, and today's hot system often doesn't last past tomorrow, but Linux, after a quarter century, is going stronger than ever. Clearly, the kernel developers are doing something right. This report provides an update on 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 the kernel 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.

> *"The Linux kernel features one of the fastest-moving development processes and involves more developers than any other open source project."*

Some development highlights since 3.18

The kernel development community remains extremely busy, as will be seen in the statistics shown below. Some of the highlights from the period since the 3.18 release on Dec. 7, 2014, include:

- Just under 115,000 changesets have been merged from 5,062 individual developers representing nearly 500 corporations (that we know about). The number of changesets (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.
- As usual, a wide array of new features has been merged during this time period. Some of the highlights include support for live patching of the kernel (allowing critical updates to be applied without rebooting or downtime), extensive support for persistent-memory devices, encrypted storage for the ext4 filesystem, adoption of the extended Berkeley packet filter (BPF) engine for in-kernel extensibility in many subsystems, security-module stacking, a mechanism for the handling of page faults in user space, numerous networking enhancements with a focus on IPv6 and data-center improvements, block-layer polling

support, copy offloading with the copy_file_range() system call, the second-generation control-group API, the OrangeFS distributed filesystem, an in-kernel tracing histogram generator, hundreds of new drivers, many thousands of fixes, and more.

- There has been an increased focus on security on a couple of levels. Support has been added for a number of hardware-based security features, including Intel's memory protection extensions and memory protection keys, and ARM's privileged execute-never mechanism. But there is also a renewed interest in hardening the kernel to prevent attackers from taking over the system even when an exploitable vulnerability is present. Much of this work is focused on bringing over ideas from the longstanding grsecurity project; some of it is funded by The Linux Foundation's Core Infrastructure Initiative.
- The kernel testing infrastructure continues to improve. The "zero-day build and boot robot" system alone found nearly 400 bugs (all of which were fixed) during this period. There is a developing self-test framework in the kernel that is continually growing in range and capability.
- The release of the 4.0 kernel, ending the 3.x series, was not indicative of anything in particular beyond the fact that the minor numbers were getting large and Linus Torvalds was "running out of fingers and toes." Every kernel release is a "major" release with significant changes; the numbering scheme no longer matters much.

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 document will concern itself with the health of the development process and where all that code came from.

Development Model

Linux kernel development proceeds under a loose, time-based release model, with a new major kernel release occurring every nine to 10 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. The relatively new Civil Infrastructure Platform project plans to maintain some stable kernels for a decade or longer.

Release Frequency

The desired release period for a major kernel release is, by common consensus, eight to 12 weeks. A much shorter period would not give testers enough time 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 on nine or 10 weeks, making kernel releases quite predictable.

This can be seen in the release history since 3.18.

Kernel Version	Release Date	Days of Development
3.19	2015-02-08	63
4.0	2015-04-12	63
4.1	2015-06-21	70
4.2	2015-08-30	70
4.3	2015-11-01	63

Kernel Version	Release Date	Days of Development
4.4	2016-01-10	70
4.5	2016-03-13	63
4.6	2016-05-15	63
4.7	2016-07-24	70

Thus, five of the releases during the period covered by this report required nine weeks, while four required 10 weeks. In some cases, the extra week was needed to track down and fix some final important bugs. More often, though, a one-week delay is the result of external scheduling factors. The 4.4 release waited an extra week to avoid opening the merge window during the holiday season, and the 10-week cycle for 4.7 happened because Linus Torvalds was on vacation. As a general rule, the kernel community has solidified its ability to deliver stable kernel releases on a nine-week cycle.

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 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 tables below:

Kernel Version	Changes (patches)
3.19	12,461
4.0	10,346
4.1	11,916
4.2	13,694
4.3	11,894

Kernel Version	Changes (patches)
4.4	13,071
4.5	12,080
4.6	13,517
4.7	12,283

The kernel community had nine busy development cycles this time around, but the patch volume fell just short of setting a record; the busiest cycle in the project's history remains 3.15, with 13,722 patches merged.

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 these tables:

Kernel Version	Changes per Hour
3.19	8.24
4.0	6.84
4.1	7.09
4.2	8.15
4.3	7.87

Kernel Version	Changes per Hour
4.4	7.78
4.5	7.99
4.6	8.94
4.7	7.31

During the period between the 3.19 and 4.7 releases, the kernel community was merging changes at an average rate of 7.8 patches per hour; that is a slight increase from the 7.71 patches per hour seen in the previous version of this report, and a continuation of the long-term trend toward higher patch volumes.

During the period between the 3.19 and 4.7 releases, the kernel community was merging changes at an average rate of 7.8 patches per hour

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

As mentioned toward the beginning of this document, 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.

The recent stable kernel update history looks like this:

Kernel Release	Updates	Fixes
3.14	74	4,833
3.19	8	873
4.0	9	757
4.1	17	1,643
4.2	8	903

Kernel Release	Updates	Fixes
4.3	5	618
4.4	16	1,892
4.5	7	973
4.6	5	550

As of this writing, the first stable update for the 4.7 release has not yet occurred.

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); thus, given that updates arrive approximately one per week, there are roughly nine updates for most kernel releases. About once each year, one release is chosen to receive updates for an extended, two-year period; as of this writing, the 3.14 and 4.4 kernels are being maintained in this manner. Kernel version 4.9 will also be maintained as a long-term, stable release.

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 81 releases incorporating 7,072 fixes.

One might wonder why Linux kernel releases require hundreds or even thousands of fixes after they are declared “finished.” In the end, 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.

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.

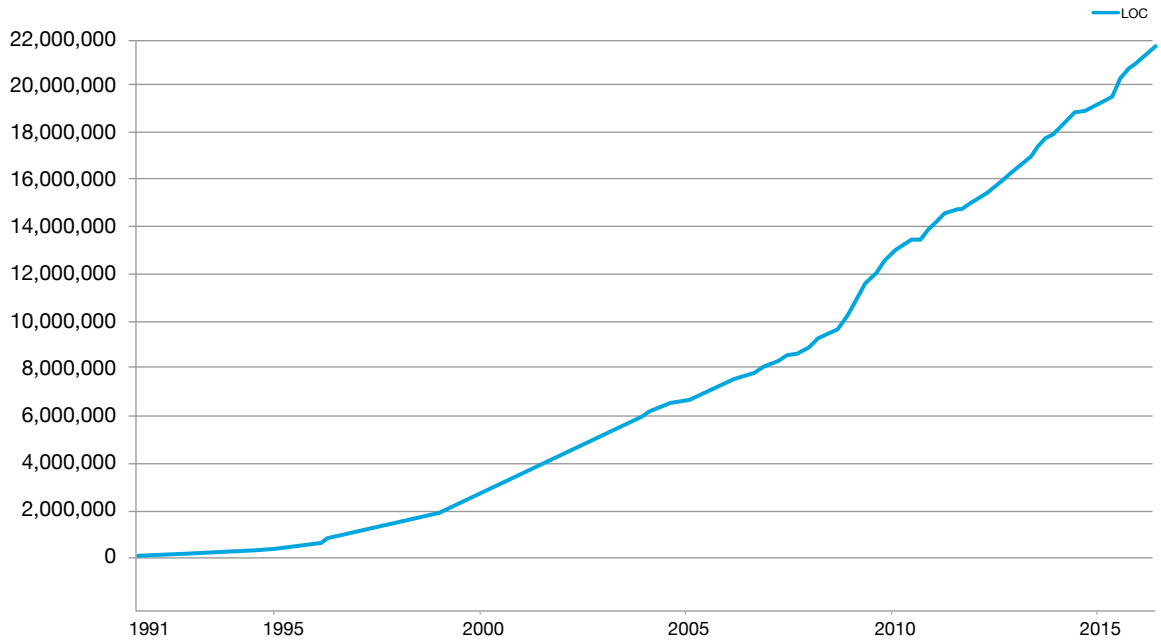
The information in the following tables shows the number of files and lines of code in each kernel version.

Kernel Version	Files	Lines
3.19	48,440	19,134,418
4.0	48,961	19,316,184
4.1	49,459	19,516,298
4.2	50,796	20,315,522
4.3	51,571	20,625,248

Kernel Version	Files	Lines
4.4	52,215	20,865,919
4.5	52,908	21,158,348
4.6	53,651	21,426,491
4.7	54,391	21,724,632

The kernel has grown steadily since its first release in 1991, when there were only about 10,000 lines of code. At almost 22 million lines (up from nearly 19 million), the kernel is almost three million lines larger than it was at the time of the previous version of this paper. Another way of putting this it that, in the production of the 3.19 to 4.7 releases, the kernel community added nearly 11 files and 4,600 lines of code — every day.

Total Lines of Code in the Linux Kernel



The kernel community added nearly 11 files and 4,600 lines of code — every day.

Who is Doing the Work

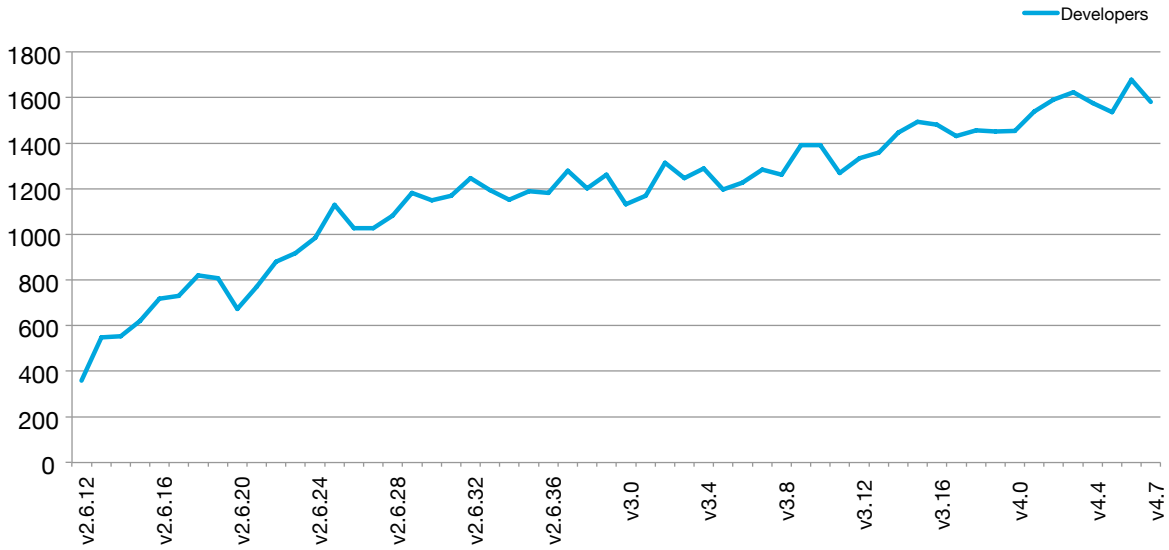
The number of different developers who are doing Linux kernel development and the identifiable companies that are sponsoring this work have been increasing over the different kernel versions, as can be seen in the following tables.

Kernel Version	Developers	Companies
3.19	1,451	230
4.0	1,458	214
4.1	1,539	238
4.2	1,591	251
4.3	1,625	211

Kernel Version	Developers	Companies
4.4	1,575	220
4.5	1,537	231
4.6	1,678	243
4.7	1,582	221

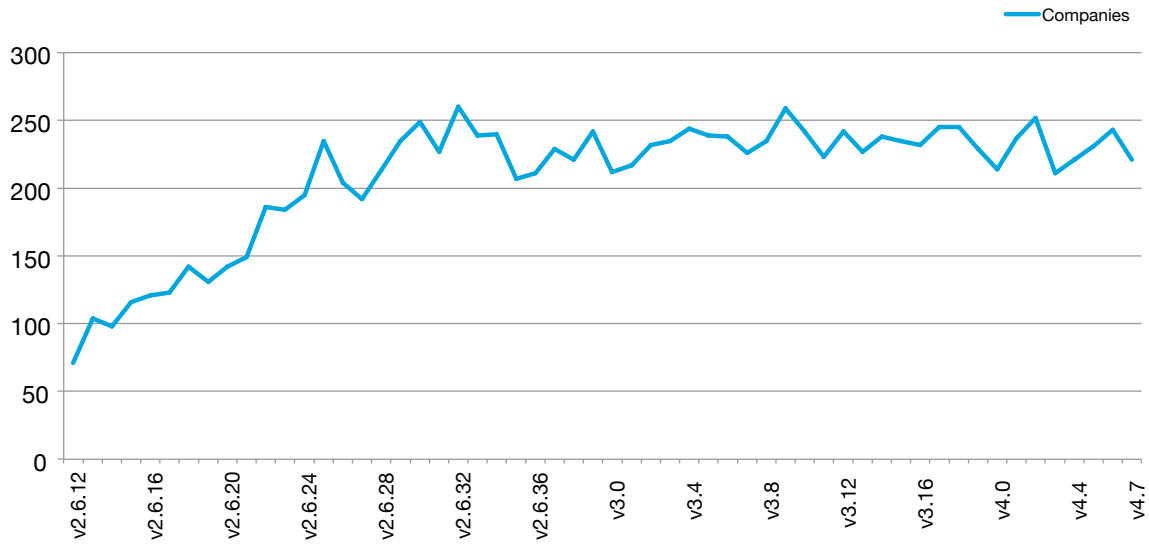
These numbers show a continuation of the steady increase in the number of developers contributing to each kernel release, with an all-time record being set with the 4.6 release.

Developers Contributing to Each Kernel Version



Since the beginning of the Git era (the 2.6.11 release in 2005), a total of 13,594 developers have contributed to the Linux kernel; those developers worked for a minimum of 1,340 companies. The number of companies supporting work on the kernel appears to be stable and not growing like the number of developers — but the 251 companies observed to have supported work on 4.2 was an all-time record.

Employers Supporting Work on Each Kernel Version



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 42,344 changes — 7.5 percent of the total. The top 30 developers contributed just over 16 percent of the total.

Those developers are:

Name	Changes	Percent
H Hartley Sweeten	5,960	1.1%
Al Viro	5,433	1.0%
Takashi Iwai	4,723	0.8%
Mark Brown	3,960	0.7%
David S. Miller	3,950	0.7%
Mauro Carvalho Chehab	3,943	0.7%
Tejun Heo	3,852	0.7%
Johannes Berg	3,707	0.7%
Russell King	3,467	0.6%
Thomas Gleixner	3,233	0.6%
Hans Verkuil	3,119	0.6%
Greg Kroah-Hartman	3,117	0.6%
Ingo Molnar	2,873	0.5%
Joe Perches	2,778	0.5%
Christoph Hellwig	2,697	0.5%

Name	Changes	Percent
Eric Dumazet	2,633	0.5%
Axel Lin	2,604	0.5%
Dan Carpenter	2,562	0.5%
Geert Uytterhoeven	2,460	0.4%
Laurent Pinchart	2,381	0.4%
Alex Deucher	2,340	0.4%
Bartlomiej Zolnierkiewicz	2,279	0.4%
Trond Myklebust	2,269	0.4%
Paul Mundt	2,268	0.4%
Daniel Vetter	2,224	0.4%
Ben Skeggs	2,216	0.4%
Arnd Bergmann	2,199	0.4%
Lars-Peter Clausen	2,176	0.4%
Arnaldo Carvalho de Melo	2,107	0.4%
Ralf Baechle	2,097	0.4%

The above numbers are drawn from the entire Git repository history, starting with 2.6.12 in 2005.

If we look at the commits since the last version of this paper (3.18) through 4.7, the picture is somewhat different:

Name	Changes	Percent
H Hartley Sweeten	1,456	1.3%
Geert Uytterhoeven	1,036	0.9%
Arnd Bergmann	877	0.8%
Al Viro	782	0.7%
Takashi Iwai	735	0.7%
Lars-Peter Clausen	729	0.7%
Mauro Carvalho Chehab	714	0.6%
Ville Syrjälä	707	0.6%
Linus Walleij	661	0.6%
Dan Carpenter	631	0.6%
Wolfram Sang	607	0.5%
Ben Skeggs	567	0.5%
Thierry Reding	563	0.5%
Daniel Vetter	557	0.5%
Krzysztof Kozlowski	548	0.5%

Name	Changes	Percent
Chaehyun Lim	545	0.5%
Eric Dumazet	543	0.5%
Christoph Hellwig	543	0.5%
Hans Verkuil	533	0.5%
Alex Deucher	493	0.4%
Thomas Gleixner	491	0.4%
Laurent Pinchart	489	0.4%
Kuninori Morimoto	488	0.4%
Arnaldo Carvalho de Melo	476	0.4%
Leo Kim	467	0.4%
Johannes Berg	459	0.4%
Mateusz Kulikowski	454	0.4%
Stephen Boyd	450	0.4%
Alexander Aring	450	0.4%
Jiri Olsa	442	0.4%

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.

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 that are employing kernel developers. For each developer, corporate affiliation was obtained through one or more of: (1) the use of company email addresses, (2) sponsorship information included in the code they submit, or (3) 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 tables below. 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 category “consultants” represents developers who contribute to the kernel as a work-for-hire effort from different companies. Some consultant companies, such as Free Electrons and Pengutronix, are shown individually as their contributions are a significant number.

The most active companies over the 3.19 to 4.7 development cycles were:

Company	Changes	Percent
Intel	14,384	12.9%
Red Hat	8,987	8.0%
none	8,571	7.7%
unknown	7,582	6.8%
Linaro	4,515	4.0%
Samsung	4,338	3.9%
SUSE	3,619	3.2%
IBM	2,995	2.7%
consultants	2,938	2.6%
Renesas Electronics	2,239	2.0%
Google	2,203	2.0%
AMD	2,100	1.9%
Texas Instruments	1,917	1.7%
ARM	1,617	1.4%
Oracle	1,528	1.4%

Company	Changes	Percent
Outreachy	1,524	1.4%
Vision Engraving Systems	1,456	1.3%
Free Electrons	1,453	1.3%
NXP Semiconductors	1,445	1.3%
Mellanox	1,404	1.3%
Atmel	1,362	1.2%
Broadcom	1,237	1.1%
NVidia	1,146	1.0%
Code Aurora Forum	1,033	0.9%
Imagination Technologies	963	0.9%
Huawei Technologies	937	0.8%
Facebook	877	0.8%
Pengutronix	790	0.7%
Cisco	692	0.6%
Qualcomm	656	0.6%

The top 10 contributors, including the groups “unknown” and “none,” make up nearly 54 percent of the total contributions to the kernel; that is down 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 80 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 paper, 13.6 percent in 2013, and 11.8 percent in 2014; over the period covered by this report, it has fallen to 7.7 percent. 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.

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.

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 (over 400 of which do not appear in the above list) which have made significant changes since the 3.19 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.

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.

Over the course of kernel development since the use of Git began, each kernel release has included contributions from 200 to 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). In the 3.x era, only 3.4 (with 182) has featured the work of less than 200 new developers.

For the time period covered by this paper, the history is:

Kernel Version	New developers
3.19	248
4.0	268
4.1	278
4.2	278
4.3	293

Kernel Version	New developers
4.4	253
4.5	219
4.6	286
4.7	232

That adds up to 2,355 first-time developers over the course of about 15 months. Remember that 4,986 developers overall contributed to the kernel during this time; one can thus conclude that nearly half 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 2,355 new developers, 55 were known to be working on their own time, while we have not yet been able to get information on 1,055 of them. The rest of the new developers (1,245 — just over half) 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:

Company	New Developers
Intel	205
Google	54
IBM	48
Huawei Technologies	44
Samsung	40

Company	New Developers
NXP Semiconductors	39
AMD	38
Mellanox	36
MediaTek	33
Red Hat	26

It is also worth noting that the Outreachy program, which helps people from underrepresented groups secure internships in free and open source software, was responsible for introducing 17 new developers to the kernel community during this time.

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.

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 3.19, the developers who added the most non-author signoff lines are:

Developer	Signoffs	Percent
Greg Kroah-Hartman	13,992	13.4%
David S. Miller	9,481	9.1%
Mark Brown	4,129	3.9%
Andrew Morton	3,654	3.5%
Daniel Vetter	2,883	2.8%
Ingo Molnar	2,622	2.5%
Mauro Carvalho Chehab	2,461	2.3%
Kalle Valo	2,205	2.1%
Arnaldo Carvalho de Melo	1,706	1.6%
Rafael J. Wysocki	1,481	1.4%

Developer	Signoffs	Percent
Ralf Baechle	1,393	1.3%
Doug Ledford	1,363	1.3%
Jeff Kirsher	1,323	1.3%
Linus Walleij	1,210	1.2%
Felipe Balbi	1,181	1.1%
Marcel Holtmann	1,171	1.1%
Jonathan Cameron	1,070	1.0%
Michael Ellerman	1,036	1.0%
Herbert Xu	986	0.9%
Jens Axboe	970	0.9%

The total number of patches signed off by Linus Torvalds (169, or 0.2% 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:

Company	Signoffs	Percent
Red Hat	19,221	18.4%
The Linux Foundation	14,180	13.5%
Intel	12,640	12.1%
Linaro	9,069	8.7%
Google	5,570	5.3%
Samsung	4,016	3.8%
none	3,835	3.7%
SUSE	3,002	2.9%
IBM	2,226	2.1%
Code Aurora Forum	2,070	2.0%

Company	Signoffs	Percent
Texas Instruments	1,911	1.8%
Facebook	1,728	1.6%
Imagination Technologies	1,434	1.4%
unknown	1,355	1.3%
Free Electrons	1,230	1.2%
consultants	1,133	1.1%
ARM	1,106	1.1%
Renesas Electronics	1,102	1.1%
University of Cambridge	1,070	1.0%
Mellanox	1,014	1.0%

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 four companies. Over the years, the concentration of subsystem maintainers has been in decline, but it is a slow process.

Lessons from 25 Years of Linux

As noted in the introduction, the kernel celebrates a quarter-century of development in 2016. Over this time, the project has gone from one strength to the next while avoiding the forks which have split the resources of competing projects. It may be many years before we fully understand the keys to the project's success, but there are a few lessons that stand out even now.

- 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, but no single company dominates kernel development. So, 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.
- Finally, the kernel shows that major developments can spring from small beginnings. The original 0.01 kernel was a mere 10,000 lines of code; now it grows by more than that every few days. Some of the rudimentary, tiny features that developers are adding now will develop into significant subsystems in the future.

Above all, 25 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.

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 the 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 that 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" (which can be found at https://www.linux.com/publications/how-participate-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/gidm.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. The analysis in this document is based on the Git contribution data alone and does not in any way aim to identify or purport copyright ownership.

The Linux Foundation promotes, protects and standardizes Linux by providing unified resources and services needed for open source to successfully compete with closed platforms.

To learn more about The Linux Foundation or our other initiatives please visit us at www.linuxfoundation.org