2020 Linux Kernel History Report

v5.8 A Publication of The Linux Foundation | August 2020

Signed-off-by: Kate Stewart <kstewart@linuxfoundation.org>
Signed-off-by: Shuah Khan <skhan@linuxfoundation.org>
Signed-off-by: Daniel M German <dmg@uvic.ca>
Reviewed-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
Reviewed-by: Jon Corbet <corbet@lwn.net>
Reviewed-by: Konstantin Ryabitsev <konstantin@linuxfoundation.org>
Reviewed-by: David A. Wheeler <dwheeler@linuxfoundation.org>
Reviewed-by: Jason Perlow <jperlow@linuxfoundation.org>
Reviewed-by: Steve Winslow <swinslow@linuxfoundation.org>
Reviewed-by: Mike Dolan <mdolan@linuxfoundation.org>
Reviewed-by: Craig Ross <ccr@linuxfoundation.org>
Reviewed-by: Alison Rowan <arowan@contractor.linuxfoundation.org>

www.linuxfoundation.org

THE LINUX FOUNDATION

Contents

Summary

With the 5.8 release tagging on August 2, 2020[1], and with the merge window for 5.9 now complete, over a million commits of recorded Linux Kernel history are available to analyze from the last 29 years. This report looks back through the history of the Linux kernel and the impact of some of the best practices and tooling infrastructure that has emerged to enable one of the largest software collaborations known. The 5.8 kernel set several records[2], so there are no signs of development slowing down.

Introduction

Over the last few decades, we've seen Linux steadily grow and become the most widely used operating system kernel. From sensors to supercomputers, we're seeing it used for launching rockets, automobiles, smartphones, watches, and many more devices in our everyday lives. Since the Linux Foundation started publishing the Linux Kernel Development Reports in 2008[3], we've observed progress between points in time.

However, we haven't been able to share the full scope of development since that first release in 1991. Since that original release, Linux has become one of the most successful collaborations in history, with over 20 thousand contributors over the 29 years, and over a million commits as of August 2020. Given the recent announcement of 5.8 release as one of the largest yet, there's no sign of it slowing down, with the latest release showing a new record of over ten commits per hour[2].

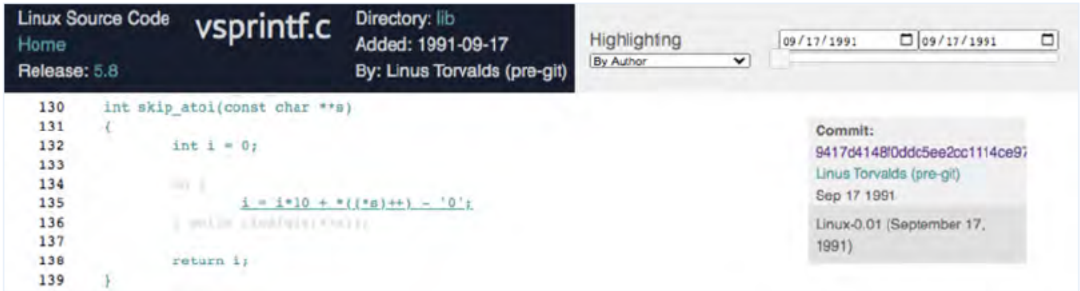
In this report, we're going to look at Linux since the first release on September 17, 1991, to August 2, 2020. This historical analysis is possible thanks to the efforts of Dr. Daniel German and his work on the tool `cregit`[4,5]. With this tool, we can look back to the first release on September 17, 1991[6], and see the kernel commits' history at the token level by contributors up to the latest 5.8 release on August 2, 2020. Using `cregit`, we have been able to connect the three different development stages of Linux development: pre-version control, `BitKeeper`, and `Git`. During the pre-version control stage, we use each of the releases of Linux as a snapshot of its development. This stage lasted from September 1991 until February 4, 2002. February 4, 2002, is when `BitKeeper` started being used and marks the start of the commit history for the Linux kernel. The current version control system, `Git`, started being used on April 16, 2005. The development histories of these three stages were connected, allowing us to track the evolution of kernel source code[7].

`Git` can track the evolution of Linux at the line level. Using `Git`, one can identify the change that introduced or last changed any line of code. Using `cregit`, it was possible to track the evolution of the kernel at the token level. A token of a source code program is the smallest individual element of a program (a token in programming is similar to the concept of word in human languages). Hence, it was possible to track, for every token in each line, who and when introduced it.

When `linux-0.01.tar.Z` kernel was released, it consisted of 88 files and 10,239 lines of code and ran on a single hardware architecture, `i386`[8]. Today, the v5.8 release consists of 69,325 files and 28,442,673 lines of code (which corresponds to **110,354,844 tokens**), and it runs on over 30 major hardware architectures[9].

Kernel Archeology

In Linus's note about the statistics on that first release[8], the initial release is summarized as "`It's 88 files with about ten thousand lines, written by yours truly except for the vsprintf routine which was co-written with Lars Wirzenius.`" When looking at the `vsprintf` file today, it's one of the few files with source code from that first commit.



Linux Source Code
Home
Release: 5.8

vsprintf.c

Directory: lib
Added: 1991-09-17
By: Linus Torvalds (pre-git)

Highlighting: By Author
09/17/1991 – 09/17/1991

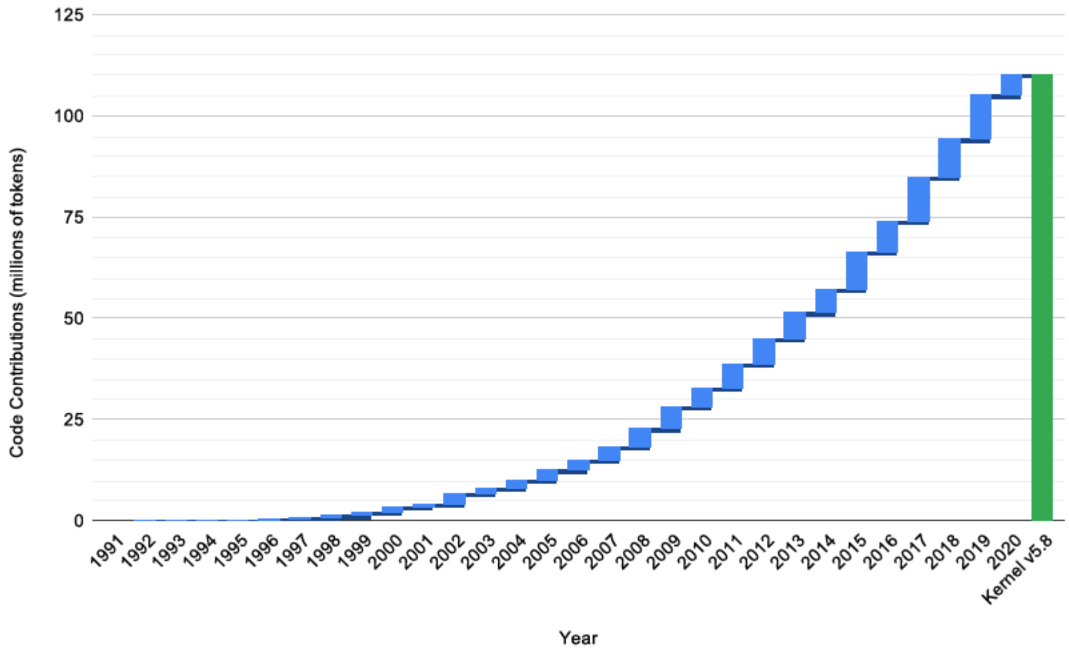
```
int skip_atoi(const char **s)
{
        int i = 0;

        do {
                i = i*10 + *((*s)++) - '0';
        } while (isdigit(**s));

        return i;
}
```

Commit:
9417d4148f0ddc5ee2cc1114ce97
Linus Torvalds (pre-git)
Sep 17 1991
Linux-0.01 (September 17, 1991)

Source: https://cregit.linuxsources.org/code/5.8/lib/vsprintf.c.html

2,964 tokens can be traced back to 1991 that can still be found in the 5.8 kernel. In fact, tokens remain in the 5.8 kernel from every year since then. Table 1 shows the breakdown of the year in which tokens were introduced that are still in the kernel today.

As Table 1 illustrates, over half of the code in the 5.8 kernel was written in the last seven years, but traces exist from all the prior years contribute to the code in version 5.8 even today.



Kernel v5.8 codebase broken down by year of code contribution

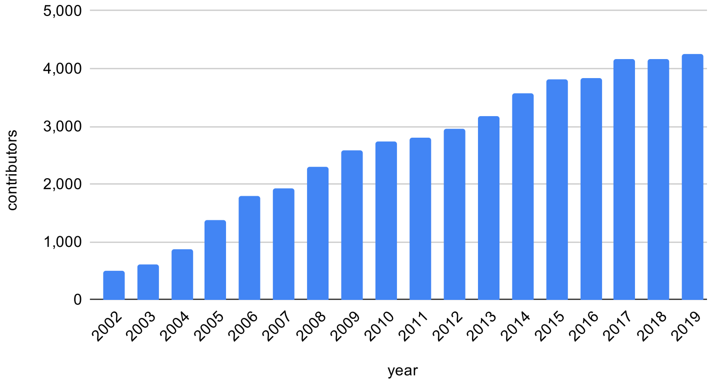
Year	# of Tokens	Proportion	Accumulated
1991	2,964	0.00%	0.00%
1992	30,740	0.03%	0.03%
1993	46,910	0.04%	0.07%
1994	38,443	0.03%	0.11%
1995	95,498	0.09%	0.19%
1996	204,573	0.19%	0.38%
1997	370,193	0.34%	0.72%
1998	436,941	0.40%	1.11%
1999	775,706	0.70%	1.81%
2000	1,469,450	1.33%	3.15%
2001	828,530	0.75%	3.90%
2002	2,475,002	2.24%	6.14%
2003	1,310,598	1.19%	7.33%
2004	2,023,705	1.83%	9.16%
2005	2,575,195	2.33%	11.49%
2006	2,449,966	2.22%	13.71%
2007	3,118,829	2.83%	16.54%
2008	4,410,921	4.00%	20.54%
2009	5,461,240	4.95%	25.49%
2010	4,805,525	4.35%	29.84%
2011	5,859,906	5.31%	35.15%
2012	6,072,494	5.50%	40.65%
2013	6,440,436	5.84%	46.49%
2014	6,091,508	5.52%	52.01%
2015	8,968,298	8.13%	60.14%
2016	7,622,786	6.91%	67.04%
2017	11,028,463	9.99%	77.04%
2018	9,248,269	8.38%	85.42%
2019	10,843,056	9.83%	95.24%
2020*	5,248,662	4.76%	100.00%

Table 1. Year of Origin of Tokens in Linux v5.8 *as of 8/2/2020

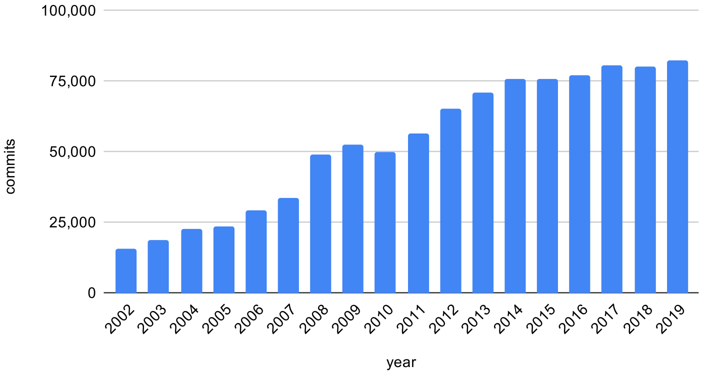
Impact of Development Process Best Practices

In 2015, The Linux Foundation's Core Infrastructure Initiative (CII) program introduced a best practices badge for open source projects to publicly determine their software development and security practices[10]. The Linux kernel was one of the first projects to get a badge. As time has evolved, additional practices have been identified, and these were incorporated into higher badge levels for projects to strive for[11]. In June of this year, the Linux kernel joined the small handful of projects with a gold badge, the top badge level[12]. This is a visible recognition of what's been there for a long time now; the Linux kernel community continues to lead on establishing best practices in the areas of software engineering and secure development at scale.

The kernel maintainers meet in person every year (except this one) at the Maintainer Summit to discuss how to improve their ways of working together. Topics for the summit are determined by developers raising them on the `ksummit-discuss` mailing list[13]. After a healthy discussion, those that aren't resolved are selected to be discussed in person. These meetings are vital to the continuous improvement of the processes that the community follows. Over time there has been a steady growth in the number of contributors and commits to the kernel each year. While it is not easy to quantify the impact of these process improvement meetings, they are likely a positive factor.



Contributors to the Linux Kernel per Year



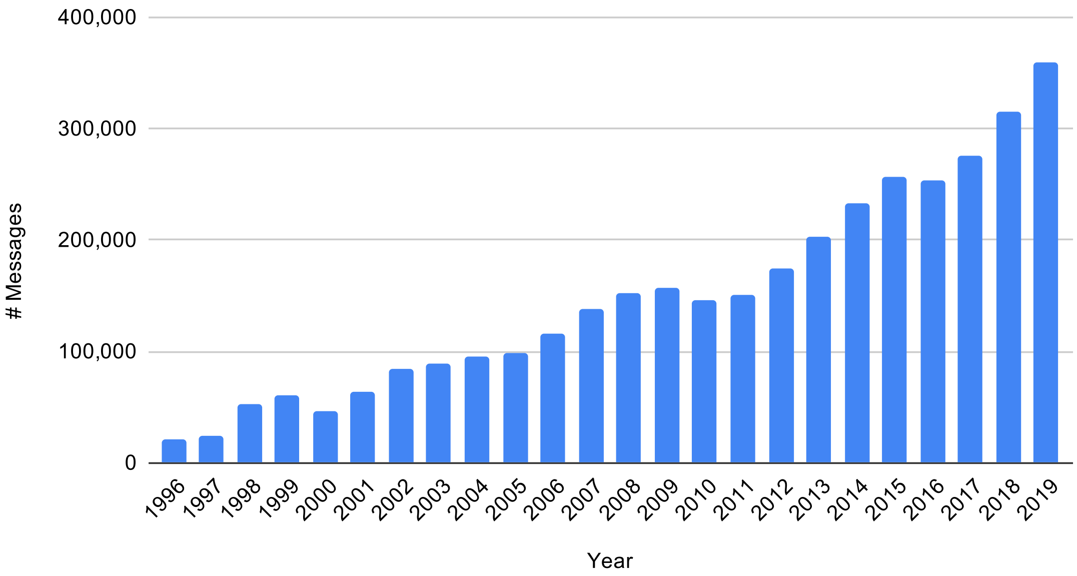
Commits to the Linux Kernel per Year

Adoption of Maintainer Hierarchy

One of the best practices that have helped the kernel scale over time is the adoption of a maintainer hierarchy. If we look back through the kernel release archives, the first instance of the `MAINTAINERS` file was committed as part of v1.3.68 in January 1996[14]. The file was only 107 lines long, and there were three maintainers listed, Alan Cox, Jon Naylor, and Linus Torvalds. That version of the `MAINTAINERS` file ends with:

```
REST:
P:     Linus Torvalds
S:     Buried alive in email
```

In trying to understand the discussions that led to the formation of this file, early Linux development mailing lists were consulted. Unfortunately, only partial records of the discussions are publicly available before 1997, as Linux development took place across multiple mailing lists and USENET groups. Before the `linux-kernel` mailing list was hosted on `vger.kernel.org`, it was at `vger.rutgers.edu`, which was only one of a number of the lists where the discussions occurred, with significant discussions still on USENET. One archive source is `http://lkml.iu.edu/hypermail/linux/kernel/`, containing `linux-kernel` mailing list ("LKML") archives going back to 1995, but even it contains some key gaps. If anyone reading this has access to early Linux discussion threads, please reach out to the `lore.kernel.org` administrators[15] so we can add more early development discussions to the archives[16].



Emails on LKML per Year

In contrast, today's v5.8 MAINTAINERS file is now 19,033 lines long and has 1501[17] maintainers listed[18]. The v5.8 MAINTAINERS file ends with:

```
THE REST
M: Linus Torvalds <torvalds@linux-foundation.org>
L: linux-kernel@vger.kernel.org
S: Buried alive in reporters
Q: http://patchwork.kernel.org/project/LKML/list/
T: git git://git.kernel.org/pub/scm/linux/
kernel/git/torvalds/linux.git
F: *
F: */
```

As the number of maintainers has increased, the number of email messages on LMKL has increased as well, but it is no longer mentioned as a problem by Linus, at least.

Version Control Systems

The version control system choice selected has had a significant impact on the community's ability to collaborate. This was a topic of much frustration in the early 2000s and spurred the break from kernel work and creation of `Git` by Linus Torvalds[19]. It's been over 15 years since `Git` development began on April 10, 2005, and it's proven to be very effective not only for the kernel community but also for many other projects.

Before `BitKeeper`'s use, we lacked transparency as to who was contributing, the number of contributions, and the paths the contributions flowed through. For this report, the commit history is considered to have started in 2002. Prior to that, we have the release history stored at Linux Kernel Historic Tree[20] and Dave Jones' `history.git`[21] to identify the earlier release information. The `CREDITS` file in v1.0 gives a listing of the early contributors, some of whom are still familiar names today[22].

There was a significant increase in the number of visible commits and the number of contributors occurring each year after the initial transition to `BitKeeper`. Then it's been steady growth after the transition to using `Git` for version control.

Since the version control system was changed from `BitKeeper` to `Git`, there's been a steady increase in the number of contributors and commits each year as well.

Year	Version Control System	Commits	Contributors
2002	BitKeeper	15,474	497
2003	BitKeeper	18,609	609
2004	BitKeeper	22,520	882
2005	BitKeeper/Git	23,553	1,372
2006	Git	29,256	1,788
2007	Git	33,761	1,928
2008	Git	48,851	2,304
2009	Git	52,576	2,593
2010	Git	49,809	2,734
2011	Git	56,405	2,806
2012	Git	65,432	2,950
2013	Git	70,966	3,167
2014	Git	75,650	3,580
2015	Git	75,827	3,817
2016	Git	77,041	3,840
2017	Git	80,826	4,175
2018	Git	80,097	4,168
2019	Git	82,371	4,249

Table 2: Commits and Contributors over time.

```
commit 1da177e4c3f41524e886b7f1b8a0c1fc7321cac2

Author: Linus Torvalds <torvalds@ppc970.osdl.org>

Date: Sat Apr 16 15:20:36 2005 -0700

Linux-2.6.12-rc2

Initial git repository build. I'm not
bothering with the full history, even
though we have it. We can create a separate
"historical" git archive of that later if we
want to, and in the meantime it's about 3.2GB
when imported into git - space that would
just make the early git days unnecessarily
complicated, when we don't have a lot of
good infrastructure for it.

Let it rip!
```

source: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?h=v2.6.12-rc2&id=1da177e4c3f41524e886b7f1b8a0c1fc7321cac2

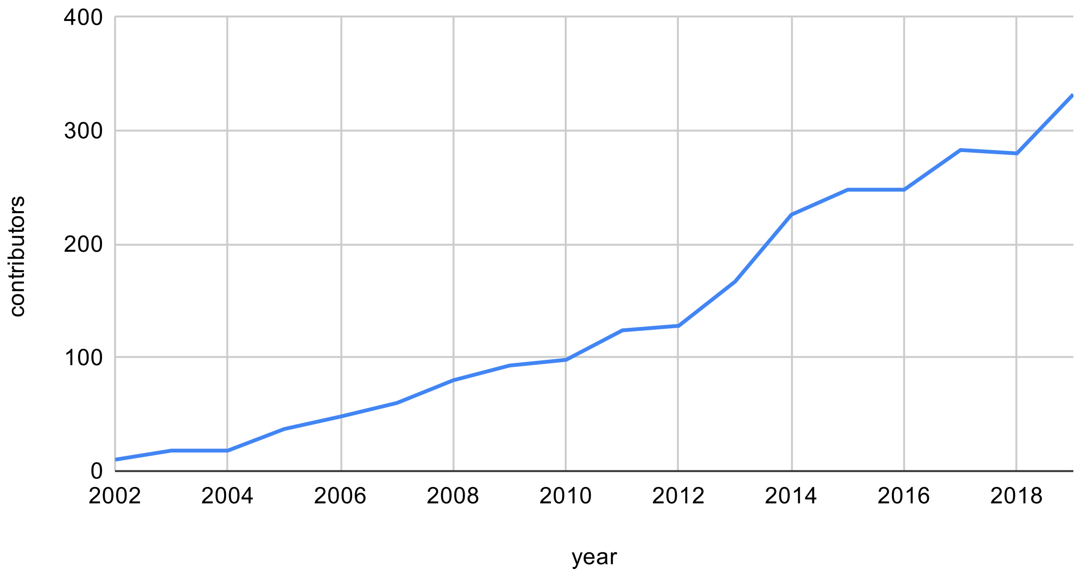
When the first kernel report was published in 2008[3], Figure 3 in that original report showed the changes per hour for the releases from 2005 to 2008. The report showed an average of 2 commits per hour for the 2.6.12 release 15 years ago in May of 2005. Fifteen years later, the average for 2019 is 9.4 commits per hour in the kernel for the last year. With the latest 5.8 kernel, the average was 10.7 commits per hour.

Having the right tool for the task has made a difference!

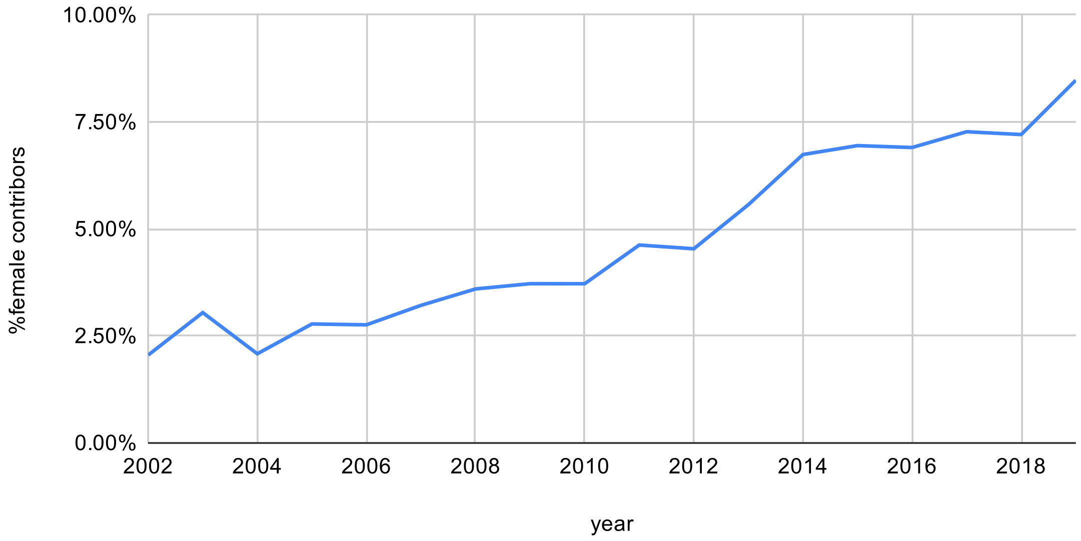
While it is not always possible to identify the gender of the contributors to the Linux Kernel, the data we have suggests a slight improvement in the diversity of contributors in recent years.

At the end of 2019, we see that 8.5% of the contributors were estimated to be female. And as a portion of the overall contributors, the percentage of female contributors has been increasing over time.

To identify the gender of contributors, we used GenderComputer[23] developed by researchers at the Eindhoven University of Technology and Carnegie Mellon University. When considered against the total contributors to the kernel though, there is still clearly room for the kernel for improving the gender diversity of participants.



Estimated Female Contributors to the Linux Kernel per Year



% of Contributors to Linux Kernel that are Estimated to be Female

The Linux Kernel community members continue to look for ways to improve the diversity of contributors, and it has been a topic at past Kernel Summits[24]. Kernel developers volunteer their own time to mentor new developers participating in programs like Outreachy[25], and the Linux Foundation's Kernel Mentorship(LKMP) program[26]. LKMP is part of the mentorship program on CommunityBridge[27] which has been helping developers new to open source to experiment, learn, and contribute to open source communities. It has ultimately helped them connect with experts in open source communities as well as employers. The program is designed to build a healthy ecosystem around the open source software we all care about and rely on by funding projects, securing code, and connecting with talented developers from diverse backgrounds.

We at the Linux Foundation have also been reaching out to new women contributors asking for suggestions for improving the materials and guidance available to new contributors. A few takeaways from the feedback:

- Online resources are inadequate or outdated and require updates.
- More FAQs on subsystems patch submission guidelines will be helpful.
- More FAQs or blogs on best practices for contributing to Linux Kernel will be helpful.
- Communication about patch acceptance status can be improved.
- More beginner-friendly courses and videos such as Shuah Khan's LFD103[28] and Greg Kroah-Hartman's "Write and Submit your first Linux kernel Patch"[29] are helpful for students with limited resources.

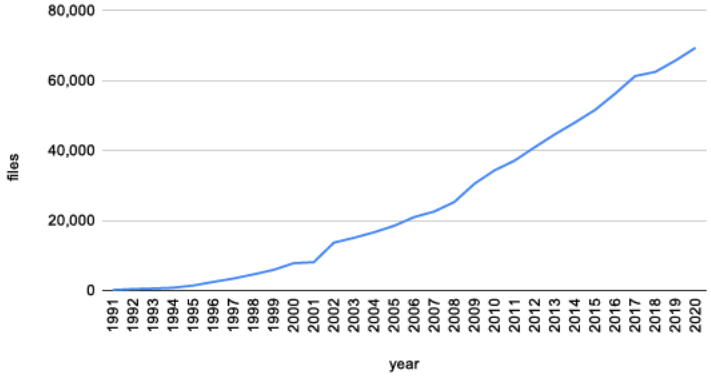
Removing Unused Code

As one would expect, as the number of commits has been increasing over time, the kernel has been growing in terms of the number of files and lines of code. It is important to remember, though, that each kernel build only incorporates a fraction of the source code's possible files. As Greg Kroah-Hartmann notes in his blog "An average laptop uses around 2 million lines of kernel code from 5 thousand files to function properly, while the Pixel phone uses 3.2 million lines of kernel code from 6 thousand files due to the increased complexity of a SoC."[30]

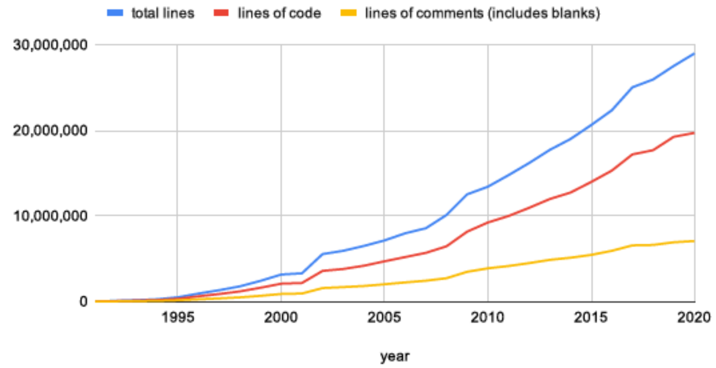
There is a continual focus to remove unused code and files, but as more features and infrastructure are added, the number of files continues to increase overall. In looking at the files in the kernel over time, there are releases (like v4.17 in 2018 where eight architectures were removed, and ~180,00 lines were removed[31]) where significant pruning is done that it can be detected in the trend lines.

In a similar manner to the files, the lines of code that make up the kernel have continued to increase. There's more than just the source code, though; comments are important for understanding as are blank lines for keeping it readable. The following table shows the lines of code, comment, and blank lines have been calculated by `cloc`[32].

From prior work done by Daniel German, each line of code is, on average, about seven tokens, and the number of lines of code tracks the total number of tokens very closely over time[33].



Number of Files in the Linux Kernel



Lines of Code & Comments

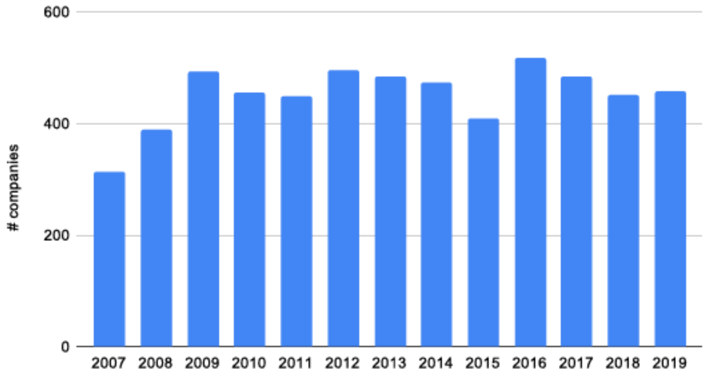
Highly Diversified Corporate Contributors

The Linux Kernel has attracted a wide range of organizations contributing to it over the years. From looking at the commit data from `Git`, we see that we have a very long tail of organizations contributing to the kernel. From 2007 to 2019, there were **780,048** commits accepted into the Linux kernel from **1730** organizations. The top 20 organizations listed below accounted for **68%** of the commits, and there is a long tail of companies that only made a single commit. The top 20 committers are at left.

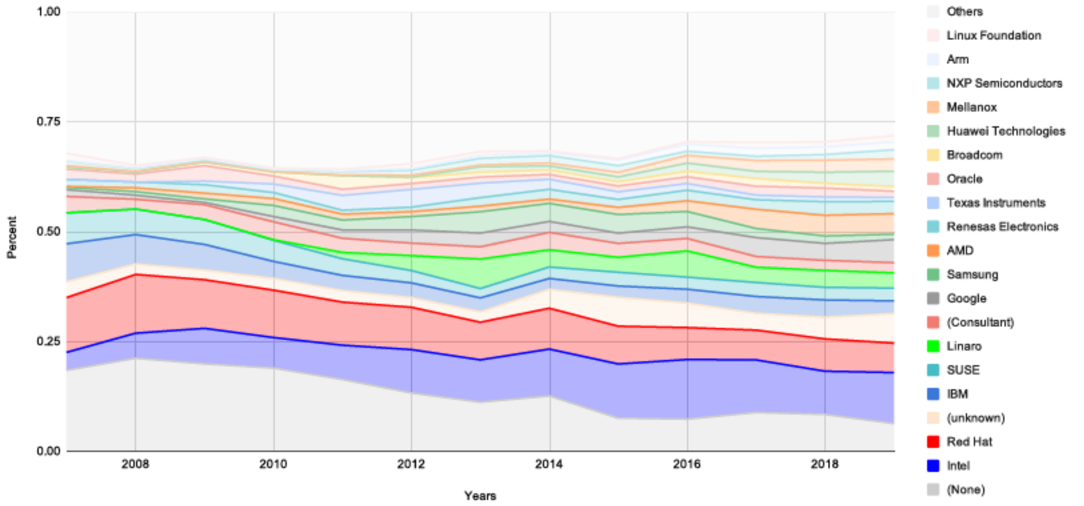
In these tables and graphs, the label of "(Unknown)" are those patches for which a supporting employer's existence could not be determined. When "None" is used, it indicates the patches from developers known to be working on their own time. This convention is used in the `gitdm` tool[34] that was used to generate these tables and was initially documented in the statistics about the 2.6.20 release[35].

Organization	# of Commits 2007-2019	%
None	93,225	11.95%
Intel	78,068	10.01%
Red Hat	69,443	8.90%
(Unknown)	31,919	4.09%
IBM	29,538	3.79%
SUSE	27,239	3.49%
Linaro	24,740	3.17%
(Consultant)	23,081	2.96%
Google	21,779	2.79%
Samsung	20,160	2.58%
AMD	17,781	2.28%
Renesas Electronics	15,542	1.99%
Texas Instruments	13,855	1.78%
Oracle	13,295	1.70%
Broadcom	9,572	1.23%
Huawei Technologies	9,379	1.20%
Mellanox	9,267	1.19%
NXP Semiconductors	9,223	1.18%
Arm	7,646	0.98%
Linux Foundation	6,109	0.78%

Table 3: Top 20 Committers



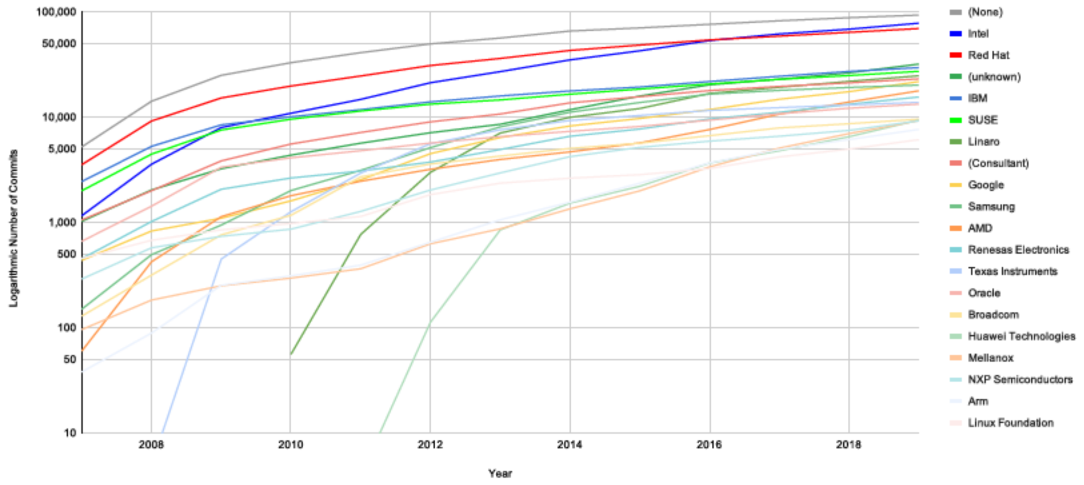
Number of Companies Committing to Linux per Year



For the last ten years, we've seen over 400 organizations contributing to the Linux kernel each year. A significant number of these may only ever submit one commit. Looking at where the percentage of each year's commits come from, long tail accounts for about one-third of all the commits (labeled as "Others" in the chart at left).

Over time, contributions from companies vary based on business needs and strategies. Some of the top 20 contributors had not even contributed to the kernel back in 2007. Others who were strong contributors in 2007, have since been bought/acquired and are no longer participating. The diversity of contributors has been a strength and continues to provide resilience to the project.

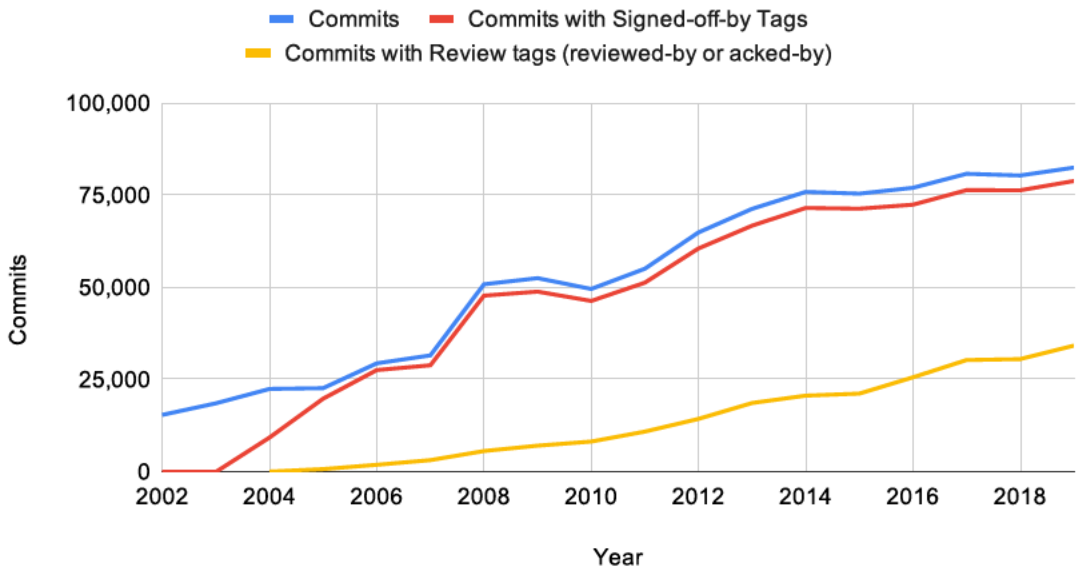
Percentage Linux Commits by Company per Year



Linux Kernel Commits by Company (cumulative)

Tagging and adoption of the Developer Certificate of Origin

One of the other key process changes in the history of the Linux kernel was the standardization of the Developer Certificate of Origin (DCO)[36] in 2004[37]. The DCO was added to provide additional legal protections to developers and users without adding a significant process burden. The DCO significantly improved the ability to track paths taken by patches to get into the kernel. Together with the transition of the version control system to `Git`, DCO has eased the overhead for developers to contribute over the last 15 years. It has proved very popular with developers and has since been adopted by other open source projects.

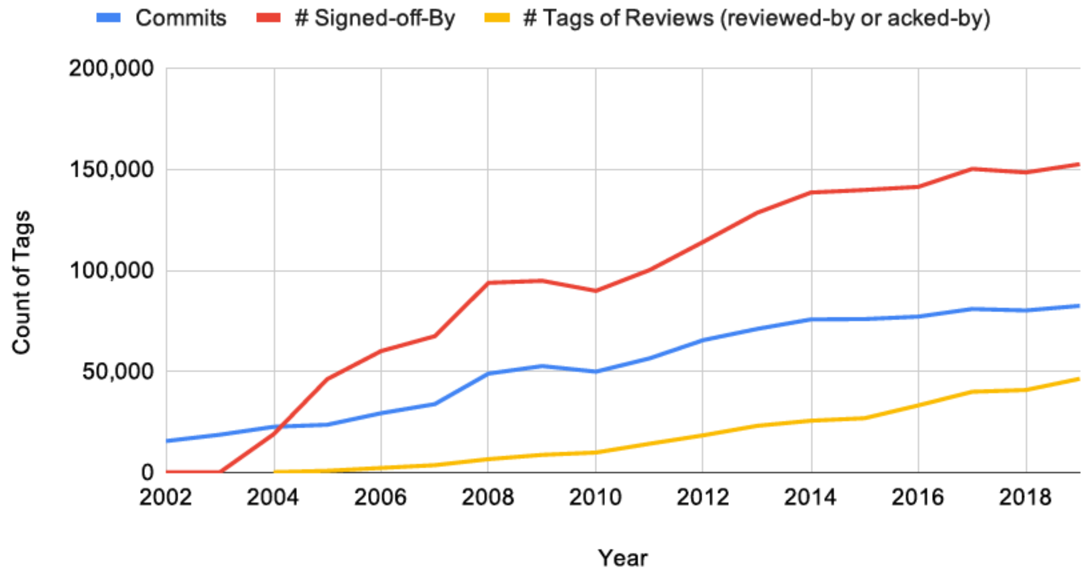


Linux Kernel Commits with Tags per Year

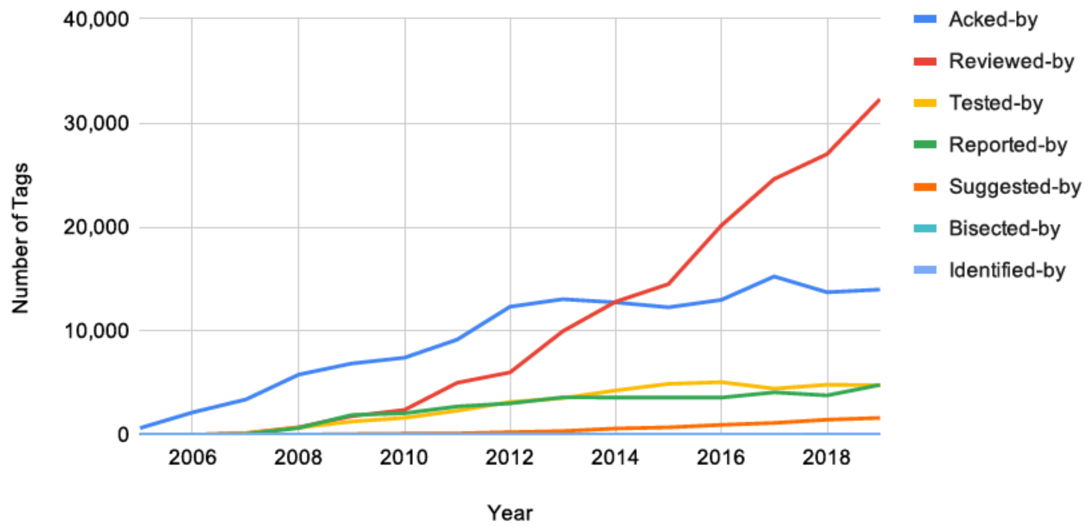
With the standardization of the use of the DCO in 2004 by the kernel community, almost all commits now have a `Signed-off-by` tag associated with them. There has also been a steady increase in visible reviews associated with each commit. The addition of tags that indicate a review (`Reviewed-by` and `Acked-by`) helps with tracking those interested in an area and have had a chance to comment and flag any concerns. Patches may need to go through several cycles of review until the relevant code maintainers have accepted the changes. This level of scrutiny before changes are committed is important to the quality of the kernel.

`Signed-off-by` tags have improved our knowledge of the paths taken by patches into the kernel. On average, there are usually two `Signed-off-by` tags associated with each commit, reflecting the hierarchy of maintainers prior to the merged code. The use of a tag to indicate a review has occurred (either `Reviewed-by` or `Acked-by`) has also steadily increased since its introduction in 2007.

However, other tags that were introduced, have not been as widely adopted and used as originally hoped. Tags do provide scope for developer creativity to be observed; some of the more interesting ones that can be found when reviewing the `git` repositories include: `“Enthusiastically-ack’d-by”`, `“Thanks-to”`, `“Based-on-the-original-screenplay-by”`, and `“Catched-by-and-rightfully-ranted-at-by”`. Moving forward, it would be great to see more use of `“Tested-by”`.



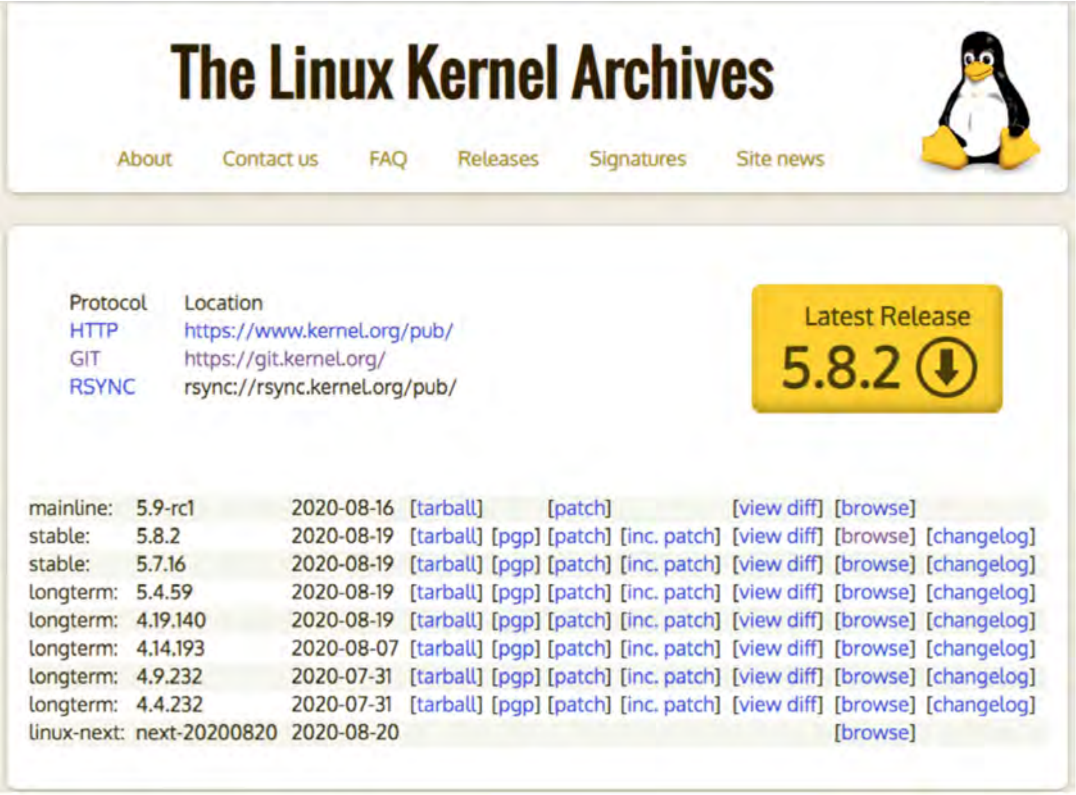
Type of Tags in the Linux Kernel per Year



Tags in the Linux Kernel per Year (excluding signed-off-by)

Release Model with Predictable Release Cycle Cadence

The Linux Kernel Release Model has evolved, and releases are considered to be classified into one of four categories: Prepatch (or "-rc") kernels, Mainline, Stable, and Long Term Stable[38]. Details and links to the active Kernel releases can be found on https://www.kernel.org/.



Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

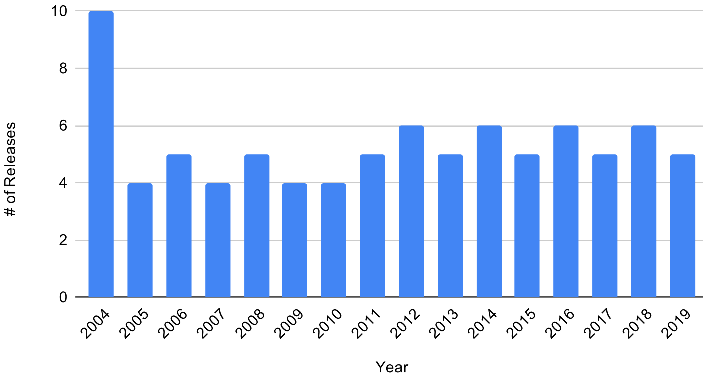
mainline:	5.9-rc1	2020-08-16	[tarball]		[patch]		[view diff]	[browse]	
stable:	5.8.2	2020-08-19	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
stable:	5.7.16	2020-08-19	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	5.4.59	2020-08-19	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	4.19.140	2020-08-19	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	4.14.193	2020-08-07	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	4.9.232	2020-07-31	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	4.4.232	2020-07-31	[tarball]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
linux-next:	next-20200820	2020-08-20						[browse]	

Left: source: https://www.kernel.org/ on 2020-08-20

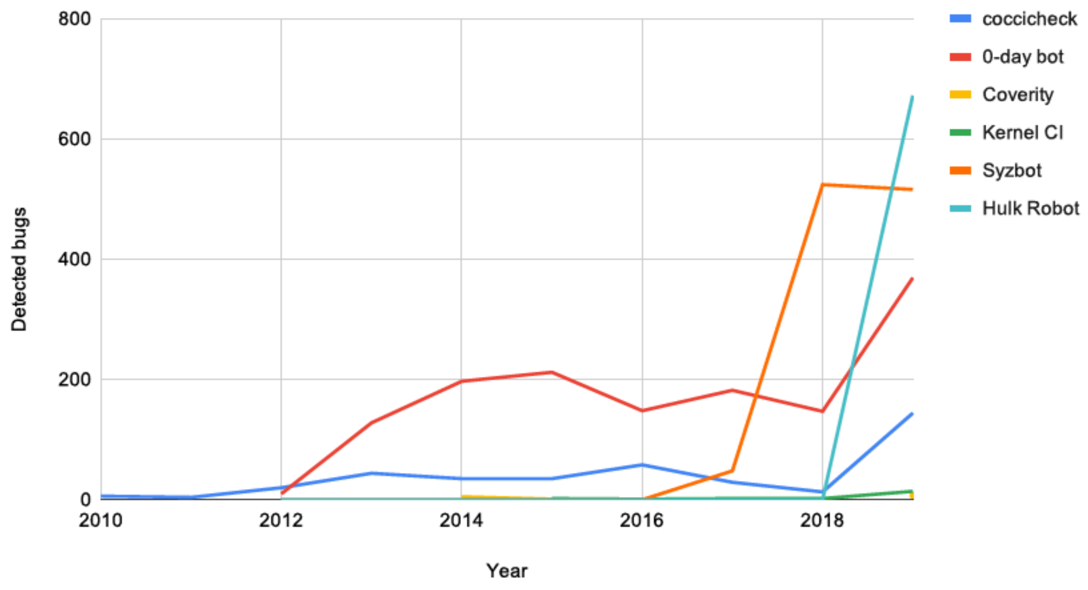
The length of the release cycle cadence has undergone much discussion and experimentation by the community. Still, since 2011 the community seems to have found a release model that works and supports the quality they seek. In fact, the release cadence has become so predictable, that a "Pointy-Haired-Boss tool"[39], was created so developers could quickly answer their corporate managers' question of "when will it be released?"

Each release cycle now starts with a two week "merge window" when new functionality can be included with the appropriate review into the git repository for the upcoming release. Once it is tagged rc1, then the cycles of integration testing, debugging, and stabilizing commence. A weekly rc candidate is then tagged until the target quality and stability is reached. The release is made, and the cycle commences again with the next merge window.

More details on the kernel release model and how it occurred can be found in Greg Kroah-Hartmann's blog on the history of how the community came up with this model[40].



Linux Releases per Year



Bots Credited with Finding Bugs in the Linux Kernel

Improving Automated Testing the Kernel

Linux Kernel testing is a community effort. Static analysis tools such as `sparse` (semantic parser)[41], `smatch` (source matcher)[42], and `coccicheck` (semantic patches that test for specific bugs)[43] are run on Linux Kernel trees by autotest bots[44] such as `0-day`[45], `Hulk Robot`[46]. In addition, fuzz testers such as Trinity: `Linux System Call Fuzzer`[47], and other tests are run to find problems proactively. Fuzz testing[48] tools such as `syzbot` are run on various kernel trees. How do these bots and tools fare finding problems? This graph shows fixes in the kernel attributed to findings by various bots, and tools.

The Linux kernel 5.4 development release statistics highlighted the impact that bots have finding bugs and being able to keep up with them[49]. The use of the "`Reported-by`" tag, attributing the bots, is helping to raise awareness of the role automation of testing is playing in preventing regressions in each release cycle.

Stable Release Process

Active stable releases come out approximately once a week. Let's talk through the release process with Linux 5.7.9 as an example. The release candidate (`rc`) is announced in an email to the stable release mailing list[50] with a pseudo short log of commits and information on where to download the release candidate from for review and testing[51]. This release candidate has 166 patches and all of these patches get sent to the mailing list in separate emails. Individual developers and automated test rings (`Kernel CI`, `0-day`, etc.) test the release candidate and report results in replying to the release candidate email. Problems found in `rc1` are fixed with subsequent `rc2`, `rc3` candidates. Release candidate testing is uneventful and releasing `rc2` is infrequent. Who does the testing?

`Buildbot` (for Linux kernel `hwmon` and `stable-queue`)[52] builds the release candidate on all 30+ supported architectures, including variants such as `arm64` and `arm` are built for multiple kernel configurations and reports results[53]. A total of 31 architectures and 56 configs are build tested during the stable release review cycle. `LKFT` (Linux Kernel Functional Testing) ring builds and tests release candidates on arm and arm64 hardware and reports results[54]. Tests run include Linux Test Project (`LTP`), Linux Kernel Selftests, Linux `Perf`, and many more. For a complete list of tests run in this ring, refer to `LKFT` Test documentation[55], and `Kernel CI`[56] ring runs boot tests on several systems and reports results.

Once the release candidate gets sufficient testing, it is released, and another email is sent out to announce the release.

Longterm Release Kernels

The introduction of Longterm release kernels, based on stable releases, has further increased Linux's popularity with product makers and increased adoption in new market segments. Greg Kroah-Hartmann wrote an excellent blog on which kernel he'd choose for a task in 2018[57], and it provides a useful overview for developers looking to select a kernel for their project today. Details of the supported long term stable releases and period until the end of the support (Projected EOL) can be found at kernel.org.

Longterm release kernels start off as a stable kernel with a commitment to maintaining for an extended period. Distributions like SUSE, Ubuntu, Red Hat, etc. helped to pioneer this concept, and it was formalized in 2010[58] with the formalization of "longterm" kernels and creation of the stable@kernel.org mailing list.

Bugs found in the current linux stable kernel are fixed upstream and then applied to the stable kernel. When a fix is determined to be applicable to one of the longterm release kernels, it is backported and applied.

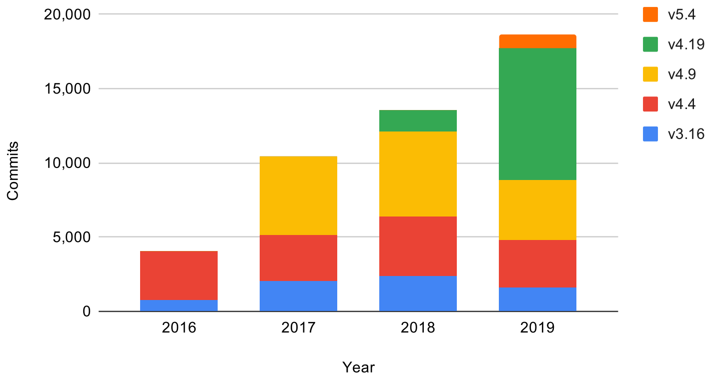
Improvements in tooling to detect when a fix can apply to a longterm stable kernel, like the work by Sasha Levin on creating a tool using machine-learning techniques to identify patches that look like useful fixes[59], have been significantly improving in recent years, and are now finding more bug fixes that can be applied to the longterm release kernels.

In 2019, 18,668 commits were made to the various LTS kernels; that's over 2.15 commits per hour. **More changes were made to the LTS releases in 2019 than to the 2.6.12 kernel, 15 years ago.**

There is market pressure for even longer support windows to support some of the applications where Linux is being used. For example, the Civil Infrastructure Platform (CIP) project[60] members have a common interest in having an infrastructure that needs to be maintained for extended periods. The developers participating in this project have decided to support 4.4 and 4.19 as Super Long-term Stable (SLTS) kernel release[61, 62]. It will be interesting to see what tools and processes these developers create to help make this vision possible in the years to come.

Version	Maintainer	Released	Projected EOL
5.4	Greg Kroah-Hartman & Sasha Levin	2019-11-24	December 2025
4.19	Greg Kroah-Hartman & Sasha Levin	2018-10-22	December 2024
4.14	Greg Kroah-Hartman & Sasha Levin	2017-11-12	January 2024
4.9	Greg Kroah-Hartman & Sasha Levin	2016-12-11	January 2023
4.4	Greg Kroah-Hartman & Sasha Levin	2016-01-10	February 2022

Source: https://www.kernel.org/category/releases.html



Tools Detect More Fixes for Longterm Linux Kernels

Conclusion

In 2020, the Linux kernel earned a gold CII best practices badge[63], which demonstrates that the project applies many practices to improve quality, improve security, and prevent defects. We see the Linux kernel now being used in products where security and safety-critical considerations are essential, from medical devices[64], to autonomous vehicles[65], and to spacecraft[66]. Improving the infrastructure so that the right level of security and safety analysis can be done confidently before using Linux in these safety-critical applications is one of the next big challenges that is being tackled[67].

The focus of the kernel community to maintain a common goal of having a high quality operating system with no regressions, willingness to create new processes and tools as needed to help them be more efficient continues to improve dependability of the Linux kernel as it gets deployed in new markets. Tooling improvements emerging in the kernel testing infrastructure are helping developers keep up with the rate change in the upstream kernel and continue to improve the stable kernels and future releases as being transparent about the processes followed. Kernel developers have demonstrated they are willing to question and discuss improvement and welcome diverse perspectives. There now exists a great foundation for the Linux kernel to continue to lead the way in creating best practices that help the entire open source software industry.

Thanks

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

The authors would also like to thank those developers who have taken the time and effort to create the infrastructure to save the history of the Linux kernel development so that others can learn from the best practices they've created.

References

[1] https://lore.kernel.org/lkml/CAHk-=wj+mDPbj8hXspXRAksh+1TmPjubc9RNEbu8EVpYyypX=w@mail.gmail.com/

[2] https://lwn.net/Articles/827735/

[3] https://www.linuxfoundation.org/publications/2008/04/linux-kernel-development-report-2008/

[4] https://cregit.linuxsources.org/

[5] https://github.com/cregit/cregit

[6] https://github.com/dmgerman/linux-pre-history/commit/9417d4148f0ddc5ee2cc1114ce97c71c5e4cb4b7

[7] https://archive.org/details/git-history-of-linux

[8] https://repo.or.cz/davej-history.git/commit/bb441db1a90a1801ef4e6546417a8d907c55d92f

[9] Architecture counts taken from boot test coverage counts from https://github.com/groeck/linux-build-test/tree/master/rootfs

[10] https://www.coreinfrastructure.org/announcements/linux-foundations-core-infrastructure-initiative-seeks-community-input-on-new-security-focused-badge-program/

[11] https://www.linuxfoundation.org/blog/2020/06/why-cii-best-practices-gold-badges-are-important/

[12] https://www.linuxfoundation.org/blog/2020/06/linux-kernel-earns-cii-best-practices-gold-badge/

[13] https://lore.kernel.org/ksummit-discuss/

[14] https://repo.or.cz/davej-history.git/commit/8bf26ec84e6362618e1abe641ac7f026c2674372

[15] https://www.kernel.org/lore.html

[16] https://korg.docs.kernel.org/lore.html

[17] `grep “M:” MAINTAINERS | sort | uniq | wc -l`

[18] https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/MAINTAINERS?h=v5.8.2

[19] https://www.linuxfoundation.org/blog/2015/04/10-years-of-git-an-interview-with-git-creator-linus-torvalds/

[20] https://git.kernel.org/pub/scm/linux/kernel/git/history/history.git/

[21] https://repo.or.cz/davej-history.git

[22] https://git.kernel.org/pub/scm/linux/kernel/git/history/history.git//?h=1.0&id=13f97bf0ff18e367f94a5ebcf6e89998ecedb80f

[23] https://github.com/tue-mdse/genderComputer

[24] https://lwn.net/Articles/662911/

[25] https://www.outreachy.org/

[26] https://wiki.linuxfoundation.org/lkmp

[27] https://communitybridge.org/

[28] https://training.linuxfoundation.org/training/a-beginners-guide-to-linux-kernel-development-lfd103/

[29] https://www.youtube.com/watch?v=LLBrBBImJt4

[30] http://www.kroah.com/log/blog/2018/02/05/linux-kernel-release-model/

[31] https://lwn.net/Articles/756031/

[32] https://github.com/AlDanial/cloc

[33] German, D.M., Adams, B. & Stewart, K. cregit: Token-level blame information in git version control repositories. Empir Software Eng 24, 2725–2763 (2019). https://doi.org/10.1007/s10664-019-09704-x

[34] git://git.lwn.net/gitdm.git

[35] https://lwn.net/Articles/222773/

[36] https://developercertificate.org/

[37] https://www.wired.com/2004/05/linux-whose-kernel-is-it/

[38] https://www.kernel.org/category/releases.html

[39] http://phb-crystal-ball.org/

[40] http://www.kroah.com/log/blog/2018/02/05/linux-kernel-release-model/

[41] https://sparse.docs.kernel.org/

[42] http://smatch.sourceforge.net/

[43] https://bottest.wiki.kernel.org/coccicheck

[44] https://bottest.wiki.kernel.org/

[45] https://01.org/lkp

[46] mailto:hulkci@huawei.com

[47] https://github.com/kernelslacker/trinity

[48] https://lwn.net/Articles/536173/

[49] https://lwn.net/Articles/804119/

[50] https://lore.kernel.org/stable/

[51] https://lore.kernel.org/lkml/20200714184115.844176932@linuxfoundation.org/

[52] https://kerneltests.org

[53] https://kerneltests.org/one_line_per_build

[54] https://lkft.linaro.org/

[55] https://lkft.linaro.org/tests/

[56] https://kernelci.org/

[57] http://kroah.com/log/blog/2018/08/24/what-stable-kernel-should-i-use/

[58] https://lwn.net/Articles/418580/

[59] https://lwn.net/Articles/789225/

[60] https://www.cip-project.org/

[61] https://www.cip-project.org/blog/2020/08/17/cip-kernel-team-helping-cip-sustain-industrial-grade-systems

[62] https://lwn.net/Articles/749530/

[63] https://www.linuxfoundation.org/blog/2020/06/linux-kernel-earns-cii-best-practices-gold-badge/

[64] https://www.techdesignforums.com/practice/technique/medical-linux-security/

[65] https://www.automotivelinux.org/about/

[66] https://www.zdnet.com/article/from-earth-to-orbit-with-linux-and-spacex/

[67] https://elisa.tech/

THE LINUX FOUNDATION

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