Linux Kernel Maintainer Statistics

April 23, 2018

As part of preparing my last two talks at LCA on the kernel community, "Burning Down the Castle" and "Maintainers Don't Scale", I have looked into how the Kernel's maintainer structure can be measured. One very interesting approach is looking at the pull request flows, for example done in the LWN article "How 4.4's patches got to the mainline". Note that in the linux kernel process, pull requests are only used to submit development from entire subsystems, not individual contributions. What I'm trying to work out here isn't so much the overall patch flow, but focusing on how maintainers work, and how that's different in different subsystems.

Methodology

In my presentations I claimed that the kernel community is suffering from too steep hierarchies. And worse, the people in power don't bother to apply the same rules to themselves as anyone else, especially around purported quality enforcement tools like code reviews.

For our purposes a **contributor** is someone who submits a patch to a mailing list, but needs a maintainer to apply it for them, to get the patch merged. A **maintainer** on the other hand can directly apply a patch to a subsystem tree, and will then send pull requests up the maintainer hierarchy until the patch lands in Linus' tree. This is relatively easy to measure accurately in git: If the recorded patch author and committer match, it's a maintainer self-commit, if they don't match it's a contributor commit.

There's a few annoying special cases to handle:

- Some people use different email addresses or spellings, and sometimes MTAs, patchwork and other tools used in the patch flow chain mangle things further. This could be fixed up with the mail mapping database that LWN for example uses to generate its contributor statistics. Since most maintainers have reasonable setups it doesn't seem to matter much, hence I decided not to bother.

- There are subsystems not maintained in git, but in the quilt patch management system. Andrew Morton's tree is the only one I'm aware of, and I hacked up my scripts to handle this case. After that I realized it doesn't matter, since Andrew merged exceedingly few of his own patches himself, most have been fixups that landed through other trees.

Also note that this is a property of each commit - the same person can be both a maintainer and a contributor, depending upon how each of their patches gets merged.

The ratio of maintainer self-commits compared to overall commits then gives us a crude, but fairly useful metric to measure how steep the kernel community overall is organized.

Measuring review is much harder. For contributor commits review is not recorded consistently. Many maintainers forgo adding an explicit ***Reviewed-by*** tag since they're adding their own ***Signed-off-by*** tag anyway. And since that's required for all contributor commits, it's impossible to tell whether a patch has seen formal review before merging. A reasonable assumption though is that maintainers actually look at stuff before applying. For a minimal definition of review, "a second person looked at the patch before merging and deemed the patch a good idea" we can assume that merged contributor patches have a review ratio of 100%. Whether that's a full formal review or not can unfortunately not be measured with the available data.

A different story is maintainer self-commits - if there is no tag indicating review by someone else, then either it didn't happen, or the maintainer felt it's not important enough work to justify the minimal effort to record it. Either way, a patch where the git author and committer match, and which sports no review tags in the commit message, strongly suggests it has indeed seen none.

An objection would be that these patches get reviewed by the next maintainer up, when the pull request gets merged. But there's well over a thousand such patches each kernel release, and most of the pull requests containing them go directly to Linus in the 2 week long merge window, when the over 10k feature patches of each kernel release land in the mainline branch. It is unrealistic to assume that Linus carefully reviews hundreds of patches himself in just those 2 weeks, while getting hammered by pull requests all around. Similar considerations apply at a subsystem level.

For counting reviews I looked at anything that indicates some kind of patch review, even very informal ones, to stay consistent with the implied oversight the maintainer's ***Signed-off-by*** line provides for merged contributor patches. I therefore included both ***Reviewed-by*** and ***Acked-by*** tags, including a plethora of misspelled and combined versions of the same.

The scripts also keep track of how pull requests percolate up the hierarchy, which allows filtering on a per-subsystem level. Commits in topic branches are accounted to the subsystem that first lands in Linus' tree. That's fairly arbitrary, but simplest to implement.

Last few years of GPU subsystem history

Since I've pitched the GPU subsystem against the kernel at large in my recent talks, let's first look at what things look like in graphics:

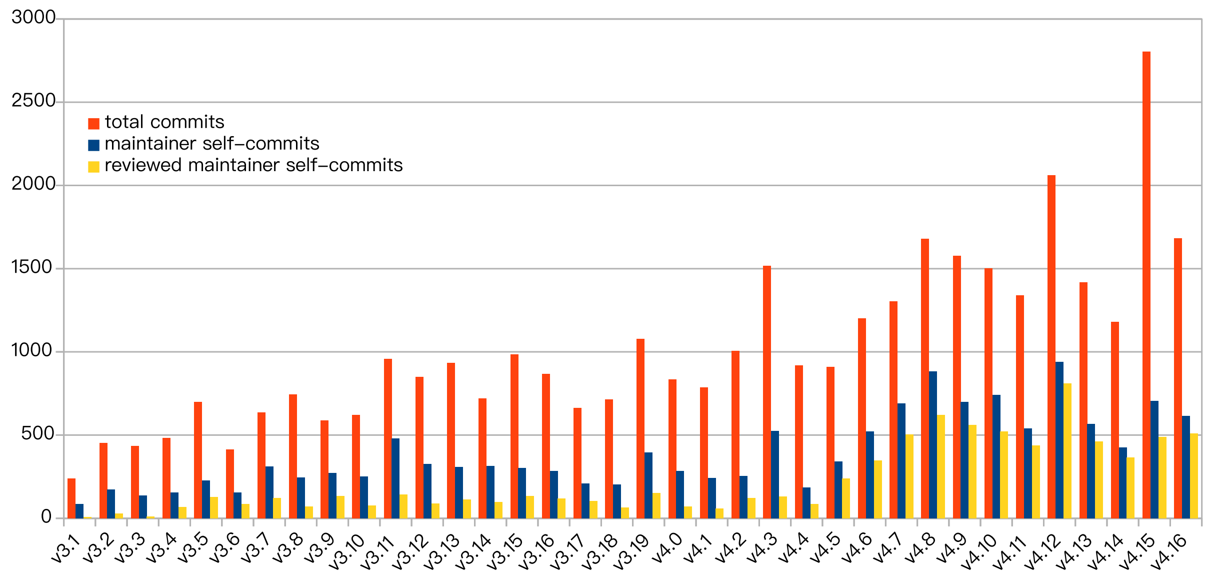

Fig. 1 GPU total commits, maintainer self-commits and reviewed maintainer self-commits

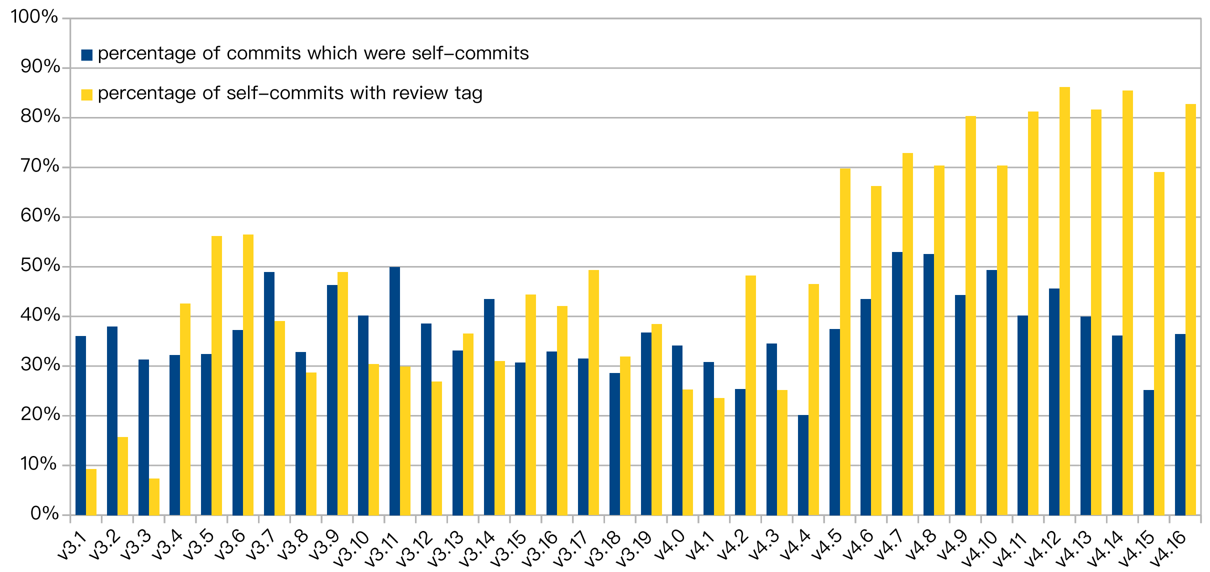

Fig. 2 GPU percentage maintainer self-commits and reviewed maintainer self-commits

In absolute numbers it's clear that graphics has grown tremendously over the past few years. Much faster than the kernel at large. Depending upon the metric you pick, the GPU subsystem has grown from being 3% of the kernel to about 10% and now trading spots for 2nd largest subsystem with arm-soc and staging (depending who's got a big pull for that release).

Maintainer commits keep up with GPU subsystem growth

The relative numbers have a different story. First, commit rights and the fairly big roll out of group maintainership we've done in the past 2 years aren't extreme by historical graphics subsystem standards. We've always had around 30-40% maintainer self-commits. There's a bit of a downward trend in the years leading towards v4.4, due to the massive growth of the i915 driver, and our failure to add more maintainers and committers for a few releases. Adding lots more committers and creating bigger maintainer groups from v4.5 on forward, first for the i915 driver, then to cope with the influx of new small drivers, brought us back to the historical trend line.

There's another dip happening in the last few kernels, due to AMD bringing in a big new team of contributors to upstream. v4.15 was even more pronounced, in that release the entirely rewritten DC display driver for AMD GPUs landed. The AMD team is already using a committer model for their staging and internal trees, but not (yet) committing directly to their upstream branch. There's a few process holdups, mostly around the CI flow, that need to be fixed first. As soon as that's done I expect this recent dip will again be over.

In short, even when facing big growth like the GPU subsystem has, it's very much doable to keep training new maintainers to keep up with the increased demand.

Review of maintainer self-commits established in the GPU subsystem

Looking at relative changes in how consistently maintainer self-commits are reviewed, there's a clear growth from mostly no review to 80+% of all maintainer self-commits having seen some formal oversight. We didn't just keep up with the growth, but scaled faster and managed to make review a standard practice. Most of the drivers, and all the core code, are now consistently reviewed. Even for tiny drivers with small to single person teams we've managed to pull this off, through combining them into larger teams run with a group maintainership model.

Last few years of kernel w/o GPU history

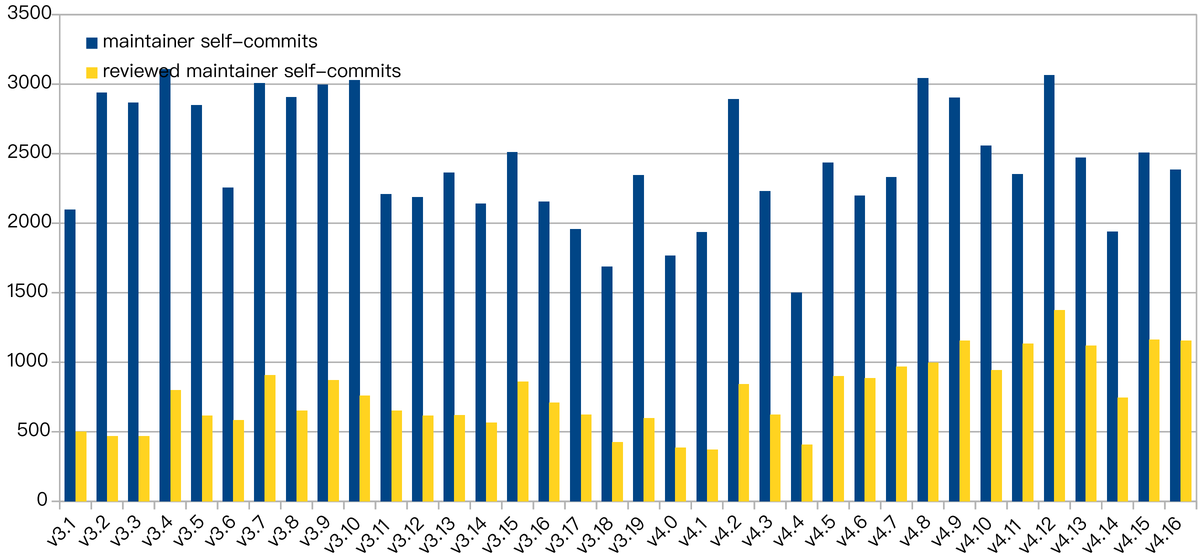

Fig. 3 kernel w/o GPU maintainer self-commits and reviewed maintainer self-commits

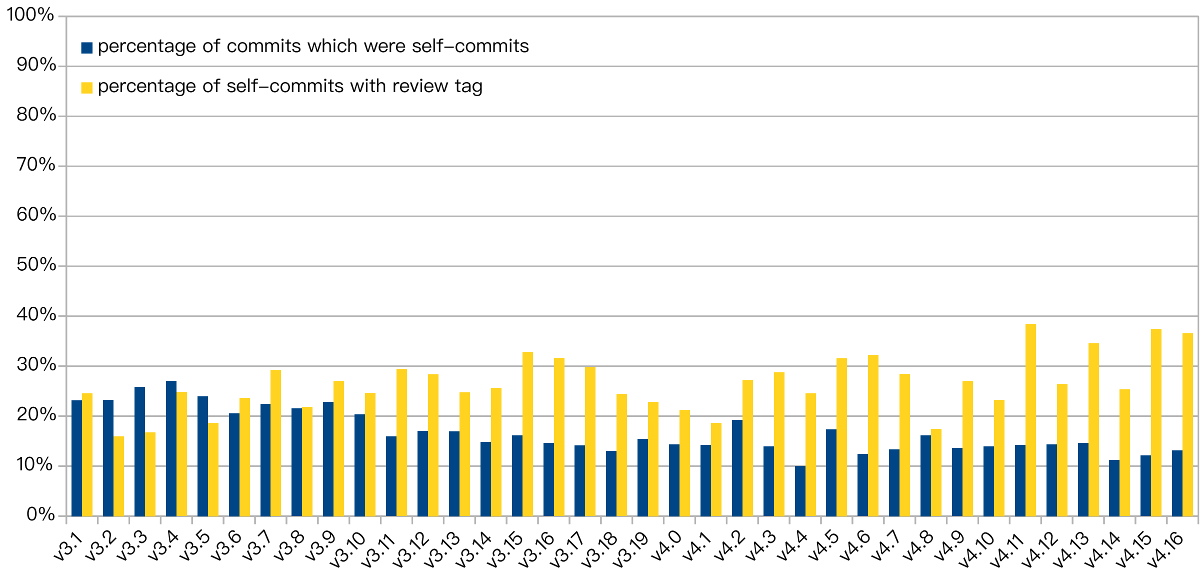

Fig. 4 kernel w/o GPU percentage maintainer self-commits and reviewed maintainer self-commits

Kernel w/o graphics is an entirely different story. Overall, review is much less a thing that happens, with only about 30% of all maintainer self-commits having any indication of oversight. The low ratio of maintainer self-commits is why I removed the total commit number from the absolute graph - it would have dwarfed the much more interesting data on self-commits and reviewed self-commits. The positive thing is that there's at least a consistent, if very small upward trend in maintainer self-commit reviews, both in absolute and relative numbers. But it's very slow, and will likely take decades until there's no longer a double standard on review between contributors and maintainers.

Maintainers are not keeping up with the kernel growth overall

Much more worrying is the trend on maintainer self-commits. Both in absolute, and much more in relative numbers, there's a clear downward trend, going from around 25% to below 15%. This indicates that the kernel community fails to mentor and train new maintainers at a pace sufficient to keep up with growth. Current maintainers are ever more overloaded, leaving ever less time for them to write patches of their own and get them merged.

Naively extrapolating the relative trend predicts that around the year 2025 large numbers of kernel maintainers will do nothing else than be the bottleneck, preventing everyone else from getting their work merged and not contributing anything of their own. The kernel community imploding under its own bureaucratic weight being the likely outcome of that.

This is a huge contrast to the “everything is getting better, bigger, and the kernel community is very healthy” fanfare touted at keynotes and the yearly kernel report. In my opinion, the kernel community is very much not looking like it is coping with its growth well and an overall healthy community. Even when ignoring all the issues around conduct that I’ve raised.

It is also a huge contrast to what we’ve experienced in the GPU subsystem since aggressively rolling out group maintainership starting with the v4.5 release; by spreading the bureaucratic side of applying patches over many more people, maintainers have much more time to create their own patches and get them merged. More crucially, experienced maintainers can focus their limited review bandwidth on the big architectural design questions since they won’t get bogged down in the minutiae of every single simple patch.

4.16 by subsystem

Let’s zoom into how this all looks at a subsystem level, looking at just the recently released 4.16 kernel.

Most subsystems have unsustainable maintainer ratios

Trying to come up with a reasonable list of subsystems that have high maintainer commit ratios is tricky; some rather substantial pull requests are essentially just maintainers submitting their own work, giving them an easy 100% score. But of course that’s just an outlier in the larger scope of the kernel overall having a maintainer self-commit ratio of just 15%. To get a more interesting list of subsystems we need to look at only those with a group of regular contributors and more than just 1 maintainer. A fairly arbitrary cut-off of 200 commits or more in total seems to get us there, yielding the following top ten list:

subsystem	total commits	maintainer self-commits	maintainer ratio
GPU	1683	614	36%
KVM	257	91	35%
arm-soc	885	259	29%
linux-media	422	111	26%

subsystem	total commits	maintainer self-commits	maintainer ratio
tip (x86, core, …)	792	125	16%
linux-pm	201	31	15%
staging	650	61	9%
linux-block	249	20	8%
sound	351	26	7%
powerpc	235	16	7%

In short there's very few places where it's easier to become a maintainer than in the already rather low, roughly 15%, the kernel scores overall. Outside of these few subsystems, the only realistic way is to create a new subsystem, somehow get it merged, and become its maintainer. In most subsystems being a maintainer is an elite status, and the historical trends suggest it will only become more so. If this trend isn't reversed, then maintainer overload will get a lot worse in the coming years.

Of course subsystem maintainers are expected to spend more time reviewing and managing other people's contribution. When looking at individual maintainers it would be natural to expect a slow decline in their own contributions in patch form, and hence a decline in self-commits. But below them a new set of maintainers should grow and receive mentoring, and those more junior maintainers would focus more on their own work. That sustainable maintainer pipeline seems to not be present in many kernel subsystems, drawing a bleak future for them.

Much more interesting is the review statistics, split up by subsystem. Again we need a cut-off for noise and outliers. The big outliers here are all the pull requests and trees that have seen zero review, not even any ***Acked-by*** tags. As long as we only look at positive examples we don't need to worry about those. A rather low cut-off of at least 10 maintainer self-commits takes care of other random noise:

subsystem	total commits	maintainer self-commits	maintainer review ratio
f2fs	72	12	100%
XFS	105	78	100%
arm64	166	23	91%
GPU	1683	614	83%
linux-mtd	99	12	75%
KVM	257	91	74%
linux-pm	201	31	71%
pci	145	37	65%
remoteproc	19	14	64%
clk	139	14	64%

subsystem	total commits	maintainer self-commits	maintainer review ratio
dma-mapping	63	60	60%

Yes, XFS and f2fs have their shit together. More interesting is how wide the spread in the filesystem code is; there's a bunch of substantial fs pulls with a review ratio of flat out zero. Not even a single ***Acked-by***. XFS on the other hand insists on full formal review of everything - I spot checked the history a bit. f2fs is a bit of an outlier with 4.16, barely getting above the cut-off. Usually it has fewer patches and would have been excluded.

Everyone not in the top ten taken together has a review ratio of 27%.

Review double standards in many big subsystems

Looking at the big subsystems with multiple maintainers and huge groups of contributors - I picked 500 patches as the cut-off - there's some really low review ratios: Staging has 7%, networking 9% and tip scores 10%. Only arm-soc is close to the top ten, with 50%, at the 14th position.

Staging having no standard is kinda the point, but the other core subsystems eschewing review is rather worrisome. More than 9 out of 10 maintainer self-commits merged into these core subsystem do not carry any indication that anyone else ever looked at the patch and deemed it a good idea. The only other subsystem with more than 500 commits is the GPU subsystem, at 4th position with a 83% review ratio.

Compared to maintainers overall the review situation is looking a lot less bleak. There's a sizeable group of subsystems who at least try to make this work, by having similar review criteria for maintainer self-commits than normal contributors. This is also supported by the rather slow, but steady overall increase of reviews when looking at historical trend.

But there's clearly other subsystems where review only seems to be a gauntlet inflicted on normal contributors, entirely optional for maintainers themselves. Contributors cannot avoid review, because they can't commit their own patches. When maintainers outright ignore review for most of their patches this creates a clear double standard between maintainers and mere contributors.

One year ago I wrote "Review, not Rocket Science" on how to roll out review in your subsystem. Looking at this data here I can close with an even shorter version:

> ***What would Dave Chinner do?***

Thanks a lot to Daniel Stone, Dave Chinner, Eric Anholt, Geoffrey Huntley, Luce Carter and Sean Paul for reading and commenting on drafts of this article.

Share this on → Twitter Google+
Tags: Maintainer-Stuff