Group maintainership models

By **Jonathan Corbet**
November 2, 2016

2016 Kernel Summit

Traditionally, kernel subsystem maintainership is a solitary job, but there has been a steady increase in the number of subsystems that are using some sort of group model instead. At the 2016 Kernel Summit, Darren Hart and Daniel Vetter talked about how these models work in practice and what their experiences might have to offer other subsystem maintainers.

Darren started by noting that there are a number of motivations behind group maintainership, starting with the fact that the work, for a busy subsystem, can often be more than one person can handle. Some sort of load balancing can help to keep maintainers from burning out. Group models are also more robust in the face of vacations, illness, or simply a day job that gets busy. Dan Williams added that group maintainership can also be a good way to develop new maintainers for the future.

There are, Darren said, two models of group maintainership seen in the kernel community. One of them is the "hands off" model, as exemplified by the arm-soc tree maintained by Arnd Bergmann and Olof Johansson. They manage a single repository, using an IRC channel to take a "lock" when they are ready to apply some changes. They maintain a log file, Olof said, so that they can always see what the other has done.

The other model is "delegation," usually seen in subsystems that use the patchwork patch-management subsystem. Patchwork can delegate the handling of each patch to a specific maintainer; Darren would like to start making more use of it. Mauro Carvalho Chehab said that this is the approach used in the media subsystem; there are two maintainers, and patches are automatically delegated by patchwork. Rafael Wysocki added that the power-management subsystem also uses it; in this case, the power-management mailing list is shared between multiple subsystems, so the automatic delegation in patchwork helps to sort out changes as they arrive.

Daniel Vetter talked a bit about the multiple-committer model used in the i915 graphics driver subsystem; it was a shortened version of the talk that was covered in this article in October. He had been working in a two-person team (with Jani Nikula) for three years, but wanted more help. He had plenty of reviewers, but couldn't find anybody else willing to be named as a co-maintainer. Patch submitters wanted to deal with the maintainer rather than with other reviewers, so he and Jani were becoming a bottleneck in the process.

In response, they decided to try out a group model where many committers have the ability to commit changes to the repository. It is generally working well, though there has been "some fallout." The way that the tree is managed, with fixes being cherry-picked into another tree, creates trouble with linux-next; they have some ideas for how to improve that interaction. Developers are also occasionally confused when a seemingly random person accepts their patches.

James Bottomley asked what the essential difference is between a committer and a maintainer in this model; Daniel answered that committers work internally, while the maintainer deals with the rest of the world. Committers in general don't want a lot of external visibility — they don't want to be listed in the MAINTAINERS file — so the solution is to call them something other than "maintainers." Ben Herrenschmidt observed that the maintainer's real job, in this model, is to accept the blame when things go wrong.

Olof asked if Daniel had observed problems with developers shopping patches around trying to find an accommodating committer. Daniel responded that, in general, he trusts his committers to say "no." There had been a couple of cases involving managers who have tried to get patches merged that way; it seems to happen once with every new manager. His response is to set up a meeting with that manager and explain how things need to be done. When asked if arm-soc had that problem, Olof responded that their model, where they deal with submaintainers rather than taking patches from developers directly, tends to keep that from happening.

The final part of the discussion centered on the workflow issues in the i915 subsystem that can cause Git to send patches multiple times — the core of the difficulty with linux-next. Daniel said that the tooling is not up to the job, but Linus responded that the workflow the group was using sounded "really nasty." What i915 is using, he said, is the submaintainer model; he should be taking pull requests from those maintainers rather than sharing a repository with them. Daniel said he is not against the submaintainer model, but it would create some coordination issues in this case; the nature of that driver (and DRM drivers in general) has a lot of developers working on the same files simultaneously.

Linus insisted, though, that, with the right habits, the submaintainer model works. Maintainers should make use of topic branches and avoid back merges with upstream trees. Daniel agreed that the i915 model would not work well for proper subsystems, but for a "leaf" like the i915 driver, it works well. The session wound down at that point.

Group maintainership models
Posted Nov 3, 2016 12:46 UTC (Thu) by **jani** (subscriber, #74547) [Link]

On the i915 workflow issues: I think the article (or, worse, the discussion!) failed to observe that the i915 multiple-committer model and the i915 fixes workflow are largely orthogonal things. They should not be conflated. Issues in one should not be used against the other.

The part that we're having issues with is the fixes workflow. We apply everything to our -next branch, and cherry-pick the fixes to a branch queued to -rc kernels. When the -next branch gets merged upstream in the merge window, the same commits sit both sides of the merge. Git doesn't handle that well, and I guess it's not unfair to call that "nasty". The workflow is not unlike the stable kernel workflow, with the important difference that stable doesn't get merged back upstream, so there are no conflicts.

The one thing I haven't seen a good answer to is this: If there's a non-rebasing -next tree, and we find out that there's a fix in there that fixes an issue in Linus' tree, what's the proper way to get that fix to -rc kernels? If you've already pushed the fix to a non-rebasing tree that can't be merged to -rc kernels, you can't merge your way out of this until the next merge window. This problem is independent of the maintainer model. We've just chosen to queue all the fixes to -next and cherry-pick from there (I can elaborate the reasons later if there's interest), but the problem doesn't go away completely by using topic branches, let alone by changing the maintainer model.

Reply to this comment

Group maintainership models
Posted Nov 3, 2016 14:21 UTC (Thu) by **dtor** (subscriber, #39360) [Link]

Can you queue your fixes to a 'current' branch, ask to pull it for -rc and merge it into your -next, resolving conflicts as needed? That will make sure git recognizes then add same commits.

Reply to this comment

Group maintainership models
Posted Nov 3, 2016 16:12 UTC (Thu) by **jani** (subscriber, #74547) [Link]

> Can you queue your fixes to a 'current' branch, ask to pull it for -rc and merge it into
> your -next, resolving conflicts as needed? That will make sure git recognizes then
> add same commits.

This assumes we always know in advance which patches we want to, and, not insifigantly, are confident to queue to -rc. Basically the decision would have to be made at the time the patch is applied. This is what we did before, with at least the following problems:

1) We would fail to identify a fix, queuing to -next instead, and being at the same situation as we are currently: either cherry-picking from branch to another, or moving from branch to another by way of rebasing.

2) We would queue a fix to -rc, only to find out a bit later that it breaks things. With the cherry-pick model, we can choose the good fixes after they've seen more testing in our -next. (This could be mitigated by moving some of our QA/CI resources from our -next to -rc, but ideally we should focus on getting our -next properly tested.)

3) With the decision of which branch to push to being so important, we often ended up in indecision and stalling while trying to always make the right decision. In the end, we'd often err on the safe side, leading to 1).

4) With our developers and QA and CI working on -next, we'd typically write the fixes on -next first, only to figure out later that the fix does not apply to -rc. We'd stall figuring out what the fix for -rc is, leaving the bug unfixed in both. Getting this right would require knowing where to push the patch before writing the patch...

5) With the fixes in -rc, and -next moving rapidly, we'd sometimes have difficult conflicts around the fixes. Now, we can always resolve to the version in our -next and be sure it contains all the fixes.

I'm sure we could find ways to mitigate all of these. It's just that as a driver, i915 really is *very* actively developed by a large crowd. See git shortlog -sn on drivers/gpu/drm/i915

> and contrast with pretty much any directory under drivers, or almost any top level directory. Our workflow of pushing everything to –next first has really worked well for us except for the problems caused by cherry–picks in git merges.

Reply to this comment