Zack's Kernel News

Chronicler Zack Brown reports on the latest news, views, dilemmas, and developments within the Linux kernel community.

By Zack Brown

The Kernel Development Process

An interesting debate recently shed some light on the kernel development process. Greg Kroah-Hartman started up the stable review cycle for the 3.3.2 kernel. The idea of the review cycle is to incorporate patches into the tree and give people a chance to test them before the 3.3.2 kernel is actually released. The goal is to get Linus Torvalds's 3.x releases as stable as possible while Linus and the rest of the developers continue preparing the next 3.x release. In theory, the 3.3.1, 3.3.2, 3.3.3, …, 3.3.*y* kernels will each be more stable and reliable than the one before.

An interesting incident this time was that several users noticed system crashes in the 3.3.1 release because of a patch dealing with the suspend/resume code. Users had found that reverting that patch would fix the problem; so, Sergio Correia and Felipe Contreras asked Greg to revert that patch in the 3.3.2 kernel. No problem, right?

Not quite. Greg replied that according to the procedures for stable kernel development, the patch couldn't be reverted until it was reverted in Linus's upstream tree as well. In fact, according to `Documentation/stable_kernel_rules.txt`, any patch submission (including a patch reversion submission) to the stable tree had to include the commit ID of the same patch in Linus's upstream version.

Felipe didn't like this answer at all and said that Greg was being too hidebound by arbitrary rules when there was a clear and obvious fix that would allow real computers to be bootable that currently weren't. He said that reverting the patch was known to produce a working system because the kernel had worked before that patch had been applied. Reverting it in Greg's 3.3.2 kernel would not be introducing any new code, but would simply be going back to a known working state.

Greg held firm on the decision though, and a number of people backed him up. Adrian Chadd from the FreeBSD project said that this particular rule was crucial for maintaining a sane development environment. Allowing the stable tree to diverge from the upstream version would create a situation in which end users and distributions would be submitting bug reports and feature requests, not to mention patches, that would only apply to the stable tree and not to the upstream version, where the real development was supposed to be taking place.

Adrian cited his own experience, saying, "We had this problem with Squid. People ran and developed on Squid-2.4. The head version of Squid-2 was stable, but that isn't what people ran in production. They wanted features and bugfixes against Squid-2.2, squid-2.4, and not Squid-2.STABLE (which at the time was Squid-2.6/Sqiud-2.7.) That … didn't work. Things diverged quite quickly and it got very ugly."

Willy Tarreau also defended Greg's decision and the general policy. He pointed out that if the developers didn't make sure that all fixes for the stable tree also appeared in the upstream kernel, the next stable tree would risk missing that fix because it would be based on the upstream tree that was also missing the fix. He said, "Most stable users will switch from a stable version to another one in a next release, and these users do not want any regression. This means that we absolutely don't want to risk that a stable version has a fix that is missing from a newer version. Yes this is a crappy and annoying process but it's the only way to ensure that fixes don't get lost during an upgrade."

Willy also pointed out that the only thing keeping the patch reversion out of the stable tree was its absence in the upstream kernel; and that this would undoubtedly be fixed soon as a result of the current conversation; after which, Greg would be free to incorporate the fix into the stable series with no problem.

Linus also got into the discussion, saying, "If -stable starts reverting things that aren't reverted upstream, what do you think happens to the *next* kernel version? We have those -stable rules for a very good reason – we used to not have them, and the above 'oops, we fixed it in stable, but the fix never made it upstream' happened *all*the*time*."

Linus also mentioned that the reversion in question was already making its way through the submission process and would get to him shortly. A little later in the discussion, he reported that the fix had made it into his Git tree and was now available for Greg to revert in the stable branch. He added, "But the important lesson to everybody should be that 'we don't lose fixes from -stable'. If a problem was found in stable, it needs to be fixed up-

ZACK BROWN

The Linux kernel mailing list comprises the core of Linux development activities. Traffic volumes are immense, often reaching 10,000 messages in a week, and keeping up to date with the entire scope of development is a virtually impossible task for one person. One of the few brave souls to take on this task is **Zack Brown**.

stream. In fact, quite often people *do* find problems in stable, because it tends to have more users more quickly than mainline. That makes it really really important to make sure that those problems get fixed upstream, and not hidden in stable due to some kind of diseased 'it's a no-op to revert it' thinking."

None of these arguments made sense to Felipe. He said he was all in favor of fixing the upstream as well as the stable kernel; he just didn't see any reason to wait on it. If there was a known bug in the stable tree, the thing to do was to fix it there. We'd still have the knowledge of the bug, and developers could fix it in the upstream tree as well. The two didn't have to be linked by formal processes.

But David S. Miller pointed out that in practice, it just didn't work that way. Before the current set of rules had been in place, fixes did go into the stable series and did get lost and not applied to the upstream kernel; sometimes, long periods would go by before the discrepancy was discovered. The rules were put in place because they forced the developers to take that extra step of ensuring that the fix went into the upstream kernel before allowing it to go into the stable tree.

This also made no sense to Felipe. He pointed out that the whole point of the stable releases was to add stability to the kernel, not to sort patches for the upstream developers. And in this particular case, he argued, real systems were unable to boot because of a known fix being left out of the stable series, just so it could be included in the upstream kernel first.

At this point, the discussion started to get repetitive. It seems to me that the spirit of Felipe's point was a good one – fixes should get into the stable series quickly and not be held up. Unfortunately, as many people pointed out, there's a practical side of things that needs to be taken into consideration, and the kernel developers have set up processes that try to address those practical concerns. In this case, for example, they decided that even though the stable series is supposed to produce stable kernels and nothing else, it just happens to also be an important and useful mechanism for improving the development kernel as well.

Therefore, a fix has to go through a bit of an extra step before it gets applied. It's not unlike a standard kernel algorithm, that does a little extra thing along its run-time path because that just happens to be a very convenient moment for that extra thing to get done. The kernel is developed at a freakishly fast pace, with huge numbers of contributors all funneling changes up to Linus, and it's this effort to treat the development process as an algorithm of its own that makes this possible.

Interestingly enough, the history of stable trees versus development trees is a perfect case in point. Originally, there was just development, with patches going into the tree and no real effort at stabilization beyond the developers' natural desire to have something that worked. Then, they implemented the old even/odd approach, wherein kernel versions *x.even* would be a stable tree and *x.odd* would be a development tree; they would each take turns, with the stable series sometimes going on for over a year before work could begin again on a new development branch.

That became tortuous, and eventually Linus abandoned that approach and went back to just developing the tree all the time again, with no particular plan for stability. At that point, folks like Greg decided to take a new approach to stable trees – namely, to take each release from Linus and make that the first release of a new stable series that would be maintained for some period of time before being dropped. At the time, Linus called these the "sucker" trees because of the insane amount of work it would require from the people maintaining them.

Over time, the various policies surrounding these trees continued to evolve and develop, addressing problems as they arose by implementing nuanced policies like the one debated in the recent thread. Eventually, the trajectories of the current policies will reveal a new problem that no one anticipated, and the algorithm of kernel development will take another evolutionary step forward. It's a fascinating and rewarding process to observe. ■■■