


LWN.net

[Content](#) ► [Edition](#) ►

User: Password: | |

Ask a kernel developer: maintainer workflow

Benefits for LWN subscribers

The primary benefit from [subscribing to LWN](#) is helping to keep us publishing, but, beyond that, subscribers get immediate access to all site content and access to a number of extra site features. Please sign up today!

In this edition of "ask a kernel developer", I answer a multi-part question about kernel subsystem maintenance from a new maintainer. The workflow that I use to handle patches in the USB subsystem is used as an example to hopefully provide a guide for those who are new to the maintainer role.

As always, if you have unanswered questions relating to technical or procedural issues in Linux kernel development, ask them in the comment section, or email them directly to me. I will try to get to them in another installment down the road.

August 22, 2012

This article was contributed by [Greg Kroah-Hartman](#).

I have some questions about what I am supposed to be doing at different points of the release cycle. -rc1 and -rc2 are spelled out in Documentation/HOWTO, and I have a decent idea that patches I accept should be smaller and fix more critical bugs as the -rcX's roll out. The big question is what do I do with all of the other patches that come at random times?

First off, thanks so much for agreeing to maintain a kernel subsystem. Without maintainers like you, the Linux kernel development process would be much more chaotic and hard to navigate. I will try to explain how I have set up my development workflow and how I maintain the different subsystems I am in charge of. That example can help you determine how you wish to manage your own development trees, and how to handle incoming patches from developers.

To answer the question, yes, you will receive patches at any point in the release cycle, but not all of them are applicable to be sent to Linus at all points in time, depending on where we are in the release cycle. I'll go into more detail below, but for now, realize that in my opinion you should not require the other developers to wait for different points in the release cycle, and, instead, you should hold onto patches and send them upstream when they are needed. I think it is the maintainer's job to do the buffering.

How best do I organize my pull-request branches so that developers know which they can pull as dependencies, and which are for-next. I don't want to over-organize it, but do want to make it easy for board submitters to test from my trees. Should my pull-request branches be long-lived, or, kill them and create new after each cycle?

It's best to stick with a simple scheme for branches, work with that for a while, and then if you find that is too limiting, feel free to grow from there. I only have two branches in my git trees, one to feed to Linus for the current release cycle, and one that is for the next release cycle. This can be seen in the [USB git tree](#) on kernel.org, which shows three branches:

- **master**, which tracks Linus's tree
- **usb-linus**, which contains patches to go to Linus for this release cycle
- **usb-next**, which contains the patches to go to Linus for the next release cycle.

Both of the usb-linus and usb-next branches are included in the nightly linux-next releases as well. That gives me and the USB developers quick feedback in case there are merge issues with other development trees or if there are build issues on other architectures that I missed.

I receive patches from lots of different developers all the time. All patches, after they pass an initial "is this sane" glance, get copied to a mailbox that I call TODO. Every few days, depending on my workload, I go through the mailbox and pick out all of the patches that are to be applied to various trees I am responsible for. For this example, I'll search on anything that touches the USB tree and copy those messages to a temporary local mailbox on the filesystem called s (I name my local mailboxes for their ease of typing, not for any other good reason.)

After digging all of the USB patches out (which is really a simple filter for all threads that have the "drivers/usb" string in them), I take a closer look at the patches in the s mailbox.

First I look to find anything that would be applicable to Linus's current tree. This is usually a bug fix for something that was introduced during this merge window, or a regression for systems that were previously working just fine. I pick those out and save them to another temporary mailbox called s1.

Now it's time to start testing to see if the patches actually apply to the tree. I go into a directory that contains my usb tree and check to see what branch I am on:

```
$ cd linux/work/usb
$ git b
master      6dab7ed Merge branch 'fixes' of git://git.linaro.org/people/rmk/linux-arm
* usb-linus  8f057d7 gpu/mfd/usb: Fix USB randconfig problems
usb-next    26f944b usb: hcd: use *resource_size_t* for specifying resource data
work-linus  8f057d7 gpu/mfd/usb: Fix USB randconfig problems
work-next   26f944b usb: hcd: use *resource_size_t* for specifying resource data
```

Note, I have the following aliases in my ~/.gitconfig file:

```
[alias]
  dc = describe --contains
  fp = format-patch -k -M -N
  b = branch -v
```

This enables me to use `git b` to see the current branch much easier, `git fp` to format patches in the style I need them in, and `git dc` to determine exactly what release a specific git commit was contained in.

As you can see by the list of branches, I have a local branch that mirrors the public versions of the usb-linus and usb-next branches called work-linus and work-next. I do the testing and development work in these local branches, and only when I feel they are "good enough" do I push them to the public facing branches and then out to kernel.org.

So, back to work. As I am working on patches [[1](#), [2](#)] that are to be sent to Linus first, let's change to the local working version of that branch:

```
$ git checkout work-linus
Switched to branch 'work-linus'
```

Then a quick sanity check to verify that the patches in s1 really will apply to this tree (sadly, they often do not.):

```
$ p1 < ../s1
patching file drivers/usb/core/endpoint.c
patching file drivers/usb/core/quirks.c
patching file drivers/usb/core/sysfs.c
Hunk #2 FAILED at 210.
1 out of 2 hunks FAILED -- saving rejects to file drivers/usb/core/sysfs.c.rej
patching file drivers/usb/storage/transport.c
patching file include/linux/usb/quirks.h
```

(Note, the 'p1' command is really:

```
patch -p1 -g1 --dry-run
```

that I set up in my .alias file years ago as I quickly got tired of typing the full thing out.)

Here is an example of patches that will not apply to the work-linus branch, but it turns out that this was my fault. They were generated against the linux-next branch, and really should be queued up for the next merge window, not for this release.

So, let's switch back to the work-next branch, as that is where the patches really belong:

```
$ git checkout work-next
Switched to branch 'work-next'
```

And see if they apply there properly:

```
$ p1 < ../s1
patching file drivers/usb/core/endpoint.c
patching file drivers/usb/core/quirks.c
patching file drivers/usb/core/sysfs.c
patching file drivers/usb/storage/transport.c
patching file include/linux/usb/quirks.h
```

Much better.

Then I look at the patches themselves again in my email client, and edit anything that needs to be cleaned up. The changes could be in the Subject, the body of the patch, or any other things that need to be touched up. With developers who send patches all the time, no changes generally need to be done in this area, but, unfortunately, I end up editing this type of "metadata" all the time.

After the patches look clean, and I've done a review of them again to verify that I don't notice anything strange or suspicious, I do one last sanity check by running the checkpatch.pl tool:

```
$ ./scripts/checkpatch.pl ../s1
total: 0 errors, 0 warnings, 73 lines checked

../s1 has no obvious style problems and is ready for submission.
```

So all looks good, so let's apply them to the branch and see if the build works properly:

```
$ git am -s ../s1
Applying: usb/endpoint: Set release callback in the struct device_type \
        instead of in the device itself directly
Applying: usb: convert USB_QUIRK_RESET_MORPHS to USB_QUIRK_RESET
$ make -j8
```

If everything built, then it's time to test the patches. This can range from installing the changed kernel and ensuring that everything still works properly and the new modifications work as they say they should work, to doing nothing more than verifying that the build didn't break if I do not have the hardware that the changed driver controls.

After this, and everything looks sane, it's time to push the patches to the public kernel.org repository, as well as notifying the developer that their patch was applied to the tree and where they can find it. This I do with [a script](#) called do.sh that has grown over the years; it was originally based on a script that Andrew Morton uses to notify developers when he applies their patches. You can find a copy of it and the rest of the helper scripts I use for kernel development in my [gregkh-linux GitHub tree](#).

The script does the following:

- generates a patch for every changeset in the local branch that is not in the usb-next branch
- emails the developer that this patch has now been applied and where it can be found
- merges the branch to the local usb-next branch
- pushes the branch to the public git.kernel.org repository
- pushes the branch to a local backup server that is on write-only media
- switches back to the work-next branch

With that, I'm free to delete the s1 mailbox, and start all over with more patches.

After this, people do sometimes find problems with patches that need to be fixed up. But, since my trees are public, I can't rebase them—otherwise any developer who had previously pulled my branches would get messed up. Instead, I sometimes revert patches, or apply fix-up patches on top of the current tree to resolve issues. It isn't the cleanest solution at times, but it is better to do this than rebase a public tree, which is something that no one should ever do.

Hopefully, this description gives you an idea how you can manage your trees and the patches sent to you to make things easier for yourself, the linux-next maintainer, and any developer who relies on your tree.

([Log in](#) to post comments)

Ask a kernel developer: maintainer workflow

Posted Aug 23, 2012 6:11 UTC (Thu) by mattsm (guest, #30416) [Link]

For me:
[alias]
bc = branch --contains

instead of

[alias]
dc = describe --contains

Ask a kernel developer: maintainer workflow

Posted Aug 23, 2012 13:36 UTC (Thu) by **zuki** (subscriber, #41808) [Link]

Why would one use patch if git am can be used? It is smarter and has better diagnostics. And git am --3way can often help if the patch was generated for an old version of the tree.

Ask a kernel developer: maintainer workflow

Posted Aug 23, 2012 20:14 UTC (Thu) by **jschrod** (subscriber, #1646) [Link]

git am is used, obviously, to apply the patch.

Greg uses patch with --dry-run to see if the patch would apply cleanly in the first place, if it should be merged to another branch, etc. Neat idea, IMHO.

Ask a kernel developer: maintainer workflow

Posted Aug 23, 2012 20:46 UTC (Thu) by **zuki** (subscriber, #41808) [Link]

Sure, but git am is sometimes able to apply the patch at a different point (with --3way). And there's no need to run with --dry-run first: git am will atomically either apply the patch, or fail. I'm not saying that using patch is wrong, just not very useful.

Ask a kernel developer: maintainer workflow

Posted Sep 5, 2012 9:44 UTC (Wed) by **jnareb** (subscriber, #46500) [Link]

Well, there is always `git apply --check`...

Ask a kernel developer: maintainer workflow

Posted Aug 23, 2012 20:51 UTC (Thu) by **gartim** (subscriber, #10123) [Link]

Thanks for writing this, helpful for an interesting programmer!

Ask a kernel developer: maintainer workflow

Posted Aug 23, 2012 23:59 UTC (Thu) by **blackwood** (subscriber, #44174) [Link]

Quick aside: I recommend git tag --contains over git describe --contains. The former dumps all tags that a patch is in and hence needs some human parsing (or a script to dig out the right linux release).

But git describe can get confused, especially if the patch is part of a -next tree that gets pulled in much later in the next kernel release, so that e.g. 3.5-rc1 is further away than 3.6-rc1 and hence git describe leads you to believe the patch was merged for 3.6 but really is part of 3.5. Happened just recently with a drm/i915 patch that Greg at first refused to apply to the 3.5 stable queue ;-)

Ask a kernel developer: maintainer workflow

Posted Aug 27, 2012 16:50 UTC (Mon) by **ecashin** (guest, #12040) [Link]

One can use the "--match" option with "git describe --contains" to filter out linux-next tags It accepted a file-globbing-type syntax last time I used it.

I think it would be "git describe --contains --match 'v3.6*'".

Ask a kernel developer: maintainer workflow

Posted Aug 24, 2012 17:46 UTC (Fri) by **travelsn** (guest, #48694) [Link]

Greg,

What mail client do you use/recommend?

Ask a kernel developer: maintainer workflow

Posted Aug 24, 2012 19:49 UTC (Fri) by **gregkh** (subscriber, #8) [Link]

Whatever works for you. Personally, I use mutt.

Ask a kernel developer: maintainer workflow

Posted Aug 25, 2012 19:40 UTC (Sat) by **cavok** (subscriber, #33216) [Link]

why patch -g1?

Ask a kernel developer: maintainer workflow

Posted Aug 25, 2012 21:53 UTC (Sat) by **gregkh** (subscriber, #8) [Link]

Long hold-over from when I used BitKeeper for kernel development way back when. It doesn't do anything for git, but it's lived in my .alias file for over a decade now that way.

Ask a kernel developer: maintainer workflow

Posted Aug 26, 2012 16:28 UTC (Sun) by **cavok** (subscriber, #33216) [Link]

Greg,

I expected also some quilt instead you use mboxes as collection point, where mutt is used to do final touches.

I've some old script of you for mutt. It takes the message and tries to make it a new patch in quilt. In which workflows do you use quilt?

Thanks.

Ask a kernel developer: maintainer workflow

Posted Aug 28, 2012 14:00 UTC (Tue) by **tdz** (subscriber, #58733) [Link]

Thanks for this very interesting article! :)