ELC Europe 2018

Getting your patches into mainline Linux

What not to do (and a few things you could try instead)

Marc Zyngier <marc.zyngier@arm.com>

October 22, 2018

Opening credits

- Been messing with the Linux kernel since 1993
  - Please blame Stéphane Eranian...
- First patches merged in 1996 (`md` driver)
  - Don’t email me if your disk array gets corrupted...
  - Offloaded maintainership in 1997
- With ARM since 2010
  - Trying to bridge architecture, hardware, and obviously Linux
- Looking after
  - KVM/arm together with Christoffer Dall
  - IRQ subsystem together with Thomas Gleixner

Disclaimer

- This isn’t a maintainer rant!
- This talk is for EVERYONE!
- Does not only apply to first time contributors
  - There is something for long time hackers as well!
- Not all maintainers will agree with me
  - If you get flamed for doing any of this, send them my way!

NO FIREARMS OR WEAPONS
PERMITTED ON THIS PROPERTY
VIOLATORS ARE CONSIDERED TRESPASSERS AND ARE SUBJECT TO FORFEITURE OR ARREST

Recurrent themes

There is a bit of a disconnect between contributors and maintainers:

Recurrent themes

There is a bit of a disconnect between contributors and maintainers:

- “My patches are being ignored”

Recurrent themes

There is a bit of a disconnect between contributors and maintainers:

- “My patches are being ignored”
- “I have posted these patches 4 times, and they are still not merged”

Recurrent themes

There is a bit of a disconnect between contributors and maintainers:

- “My patches are being ignored”
- “I have posted these patches 4 times, and they are still not merged”
- “I’ve copied this code from a mainline driver, and you’re telling me it isn’t right”

Recurrent themes

There is a bit of a disconnect between contributors and maintainers:

- "My patches are being ignored"
- "I have posted these patches 4 times, and they are still not merged"
- "I've copied this code from a mainline driver, and you're telling me it isn't right"
- "I only want this code merged, I don't have the time to do all this extra work"

Recurrent themes

There is a bit of a disconnect between contributors and maintainers:

- "My patches are being ignored"
- "I have posted these patches 4 times, and they are still not merged"
- "I've copied this code from a mainline driver, and you're telling me it isn't right"
- "I only want this code merged, I don't have the time to do all this extra work"
- "But I'm giving you this code *for free*, why aren't you just taking it?"

Recurrent themes

There is a bit of a disconnect between contributors and maintainers:

- “My patches are being ignored”
- “I have posted these patches 4 times, and they are still not merged”
- “I’ve copied this code from a mainline driver, and you’re telling me it isn’t right”
- “I only want this code merged, I don’t have the time to do all this extra work”
- “But I’m giving you this code *for free*, why aren’t you just taking it?”
- “I don’t have the time to understand this, just tell me what I should write”

The characters

The Contributor

- Submits a change to the mainline kernel source
- Intends to get it merged eventually
- Can be
  - a new feature
  - a bug fix
  - code cleanup
- Complexity of the changes ranges from trivial to brain-melting
  - The contributor is often the one who understands the problem best

The Reviewer

- Can be distinct from the maintainer
- Often other contributors
- Spreads the load, so that maintainers do scale
- Probably the least recognised, and yet one of the most important characters in this story

The Maintainer

- The maintainers are responsible for some piece of code in the kernel:
  - not to break
  - to be secure
  - readable, understandable
- Ultimately the ones who put their neck on the line
- Spend an awful lot of time reviewing other people’s code
  - Often the target of hundreds of emails a day

Motivations

These characters have quite a few things in common:

- Meeting at a single point of contention: the code
- Trying to solve difficult problems
- Individual responsibility, personal investment
- Very often not their main job
- Quite often a contributor grows into a reviewer, and then a maintainer

The plot

The kernel submission workflow

You have written patches for a wicked idea:

- Post a patch series
- Get it reviewed
- Respond to comments
- Rinse, repeat

Looks simple, but there is a lot behind this.

The kernel submission workflow

You have written patches for a wicked idea:

- Post a patch series
- Get it reviewed
- Respond to comments
- Rinse, repeat

Looks simple, but there is a lot behind this.

- What is that “patch series” thing?
- Who do I send it to? How do I get it reviewed?
- I don’t understand these comments and other requests
- ...

The kernel submission workflow

You have written patches for a wicked idea:

- Post a patch series
- Get it reviewed
- Respond to comments
- Rinse, repeat

Looks simple, but there is a lot behind this.

- What is that “patch series” thing?
- Who do I send it to? How do I get it reviewed?
- I don’t understand these comments and other requests
- Can be overwhelming

What is a patch series

- It is an ordered set of patches
- It is conceptually a single change
- Split into multiple patches
- Splitting patches is a hard topic
- Nothing in the kernel breaks at any point in the middle of the series
- We have a limited capacity to process huge changes in one go

What does a patch series look like

- Each patch has a title and a clear commit message
- Each patch is numbered x/n (patch number x out of n)
  - Where x is unique, n is constant across the series, and x <= n
- It has a unique version number for the whole series
  - Do not post a series with the same version number twice!
- It has a cover letter, numbered 0/n
  - Usually only if there is more than a single patch
  - The cover letter describe the goal of the series and contains a change log
  - It contains a diff-stat of the whole series
  - All the patches in the series are in reply to the cover letter

Why these requirements?

From a maintainer or reviewer point of view, these requirements are crucial:

- Ordered:
  - Allows the reviewer to see a progression in the design
  - Needed for bisection
- Logical changes:
  - Multiple things changing at once make things hard to review
- Patch numbering:
  - Am I missing any patch in this series?
  - Helps with the ordering/threading in an email client
- Version numbering:
  - Is this something new? Or has it been reviewed already?
  - Don’t reply with a single patch with a new version number
- Cover letter:
  - So you know what changed from one revision to another
  - Make sure all the recipients of the series receive the cover letter
  - A chance to having a conversation with the maintainers

Patch series: Don’t do that

- If you’re about to send something that may end up looking like this:

```
(Mon)18:58 [ Anonymous ] RESEND [PATCH v5 10/12] arm64: vdso: replace gettimeofday.S with global vgettimeofday.C
(Mon)18:58 [ Anonymous ] RESEND [PATCH v3 1/3] arm64: compat: Split the sigreturn trampolines and kuser helpers (C sources)
(Mon)18:58 [ Anonymous ] RESEND [PATCH v3 2/3] arm64: compat: Split the sigreturn trampolines and kuser helpers (assembler ...
(Mon)18:58 [ Anonymous ] RESEND [PATCH v3 3/3] arm64: compat: Add CONFIG_KUSER_HELPERS
(Mon)18:58 [ Anonymous ] RESEND [PATCH] arm64: compat: Expose offset to registers in sigframes
(Mon)18:58 [ Anonymous ] RESEND [PATCH v2 6/6] arm64: Wire up and expose the new compat vDSO
```

- ... please don’t.
- Trying to make sense of this series is just too hard
- Probably missing on some very good code
- This is a net loss for the kernel

Use the tools, Luke

- `git` is really the only tool you need
  - and there is no life worth living outside of `git`…
- Although you can use some tool on top of `git` itself
  - But really, you don't need that
- Do not send patch series by hand. Ever.
- One-off configuration:
  - Configure `git` as an email client
  - Set `sendmail.tocover=1`, `sendmail.cccover=1` in `~/.gitconfig`
- For each series you want to send:
  - Identify the recipients for this series. Use `scripts/get_maintainers.pl`
  - `git format-patch -o patches/blah -v3 --cover-letter v4.19..HEAD`
  - Edit `patches/blah/v3-0000-cover-letter.patch`, adding the recipients in your cover letter
  - `git send-email --dry-run patches/blah/v3-*patch`
  - If it looks good, drop the `--dry-run` and let it roll

Using email

Please use the canonical email etiquette when posting or responding:

- Plain text email only, no HTML
- Reply inline, not top-posting
- Avoid attachements if at all possible
- No silly disclaimer (this is a public mailing list!)
- Cc people when it matters
- Keep the Cc list short
- Trim the email you’re responding to the essential context

Digression: why email

"Why do you use this silly outdated technology instead of `[web-thing-of-the-week]`?"

Well, email is:

- Multi-platform
- Archived
- Available offline
- Not interactive
- Distributed
- Easy to integrate with `git` and CI

Getting rid of email would require a new system to satisfy these properties.

Of course, quite a few organisation cannot do email properly...

- That's a valid concern
- A lot of people are using their personal email for this
- ELC talk idea for next year: ***SMTP in a ~~hostile~~ corporate environment***

Of reviewers and bandwidth

- You've posted a patch series two days ago and quickly received some comments
- You've quickly addressed those, collected all the Acks and review tags
- ... and now eager post a new version

Now take a deep breath. Give a chance to other reviewers to catch up with your work.

- Posting too often is usually counter-productive
- Only results in a DoS on the reviewer (you don't want that)
- Allow about a week between each version, unless asked for an immediate respin
- Remember how long it took you to write these patches
- Reviewing them won't be any quicker

Screenplay

The maintainer/reviewer workflow

- Is it something I’m interested in or maintain?
- Does the patch series make sense?
- Is there any reported failure?
- Fix or feature?
- Prioritisation
- Each maintainer or reviewer has specific requirements
  - No such thing as *One Size Fits All*

The maintainer/reviewer workflow

- Is it something I’m interested in or maintain?
- Does the patch series make sense?
- Is there any reported failure?
- Fix or feature?
- Prioritisation
- Each maintainer or reviewer has specific requirements
  - No such thing as *One Size Fits All*

But there is something that influences the above: **Trust**

- This is how we recognise contributors and reviewers
- Most often people who *go the extra mile*
- When a reviewer or maintainer asks for some extra work on a patch series
  - It is not to annoy the contributor
  - It is to improve the overall quality of the kernel itself
  - Eventually to build trust between the two parties

Why trust?

This is how the overall patch merging model works. There is trust between:

- Linus and the top-level maintainers
- top level maintainers and their sub-maintainers
- co-maintainers of a single subsystem

In the end, this trust is just as important as the code.

- A software project that doesn't encourage contributions dies
- One of maintainers' role is to retain the best contributors
- We always need new reviewers and co-maintainers
- The best contributions benefit the largest part of the community
- This requires involvement of all parties

Building trust

A maintainer or reviewer can ask you to do some additional work:

- Provide a better infrastructure
- Refactor code to limit duplication
- Move bits of a feature to core code

Try to step away from your own code for a while...

- See how this request fits into the overall kernel
- If the request is unclear, ask for clarification!
- If you think this isn't justified, try to come up with your own proposal
- The maintainer is not always right, give them an alternative perspective
- Become the trusted maintainer of your own code!

Digression: Drive-by patching

- One-off contributions
  - Contributor never to be seen again
- Sometimes the kernel equivalent of “fly-tipping”
  - See the above *“Just take it already”*
- We do not want to discourage this
  - A number of bug fixes come from those one-offs
  - Some other are just a bunch of unmaintainable changes
- We’d also like to convince these people to stick around...
  - After all, we all started with this first patch...

It is unclear how we can incentivise these contributors to

- Look for another issue to fix
- Have a more continued engagement with the kernel community

Becoming a reviewer

One of best way to improve your kernel-foo is to review patches

- Pick something you're interested in
- You don't have to be an expert in the domain
- You just need to be able to follow the code
- If something seems unclear, ask questions!
- If you spot a problem, say so!

If you're satisfied with the way the code looks:

- Optionally provide a "`Reviewed-by`"
- Remember that you're reviewing "to the best of your ability"
- Even if you're not providing a tag, your input is valuable

Be your first reviewer

Before you're about to send a patch series:

- Read your own patches
  - I mean it!
  - Really!
  - This is the best way to catch basic mistakes
- Put yourself in the reviewer's shoes
  - Does this code make sense?
  - Is it split in a coherent way?
  - Is it commented, documented well enough?
  - Have you taken all the review items into account?
  - Have you collected all the `Acked-by:` and `Reviewed-by:` tags?
- If you've answered "yes" to all the above, ship it!

Closing comments

- Contributing to the Linux kernel is both tough and rewarding
- We are all trying to work together on changing some part of a code-base
- Understanding each other’s point of view is key – but can be really hard
- Building a level of trust and understanding makes everything easier
- We have tools and processes for good reasons – we are not just trying to be difficult. Honestly.
- Ask me anything if you’re in doubt. Please trust me to be friendly.

arm

Thank you