Kernel maintainership: an oral tradition – PRELIMINARY VERSION –

Gregory CLEMENT
Free Electrons
gregory.clement@free-electrons.com

(Image credit: Andrew Cheal under license CC BY-ND 2.0)

Gregory CLEMENT

- Embedded Linux engineer and trainer at **Free Electrons**
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
- Open-source contributor
  - Contributing to **kernel support for the Armada 370, 375, 38x, 39x and Armada XP** ARM SoCs from Marvell.
  - Co-maintainer of mvebu sub-architecture (SoCs from Marvell Engineering Business Unit)
  - Living near **Lyon**, France

Free Electrons
Linux Experts

Motivation and Overview

- Motivation
  - Implicit or unwritten rules.
  - Make such rules more explicit.
  - Help new maintainers and contributors.
  - Guideline I would have liked to find.
- Overview
  - The role of a maintainer
  - Accepting a patch
  - Interaction with other maintainers

The role of a maintainer

- Gathering patches for the subsystem
  - Through emails.
  - Sometime through a git tree.
- Reviewing the submitted patches
  - Best case: accepted as is.
  - Most often: ask for a new version pointing the part to improve.
  - Worst case: rejected.
- Pushing the gathered patch to the upper subsystem
  - Pull request to another maintainer.
  - Or directly to Linus Torvalds.

Becoming a maintainer

- Creating a new subsystem:
  - Most obvious.
  - Under `arch/` usually a new family of a **CPU** or an **SoC**.
  - Under `driver/` usually a new **framework** or a **specialization** of an existing class driver.
- Joining the current maintainer:
  - After being active in the subsystem especially by doing review.
  - Generally asked by the current maintainer(s) but sometime after offering the help.
- Replacing a maintainer:
  - Either co-opt by the current maintainer before leaving.
  - Or asked by upper maintainer because of your involvement in this subsystem.
  - Or on a volunteering base often because you need to push your own patches.

Expectation of the submitter

- Reviewing the patch in a couple of days (or hours)
  - Writing and testing the code took a long time, reviewing it would be fast.
  - Eager to have a feedback to make things move on.
- Knowing the hardware by heart
  - As maintainer of the subsystem you appear as the expert of the hardware it supports.
  - You supposed to have all the variant of the hardware.
- Updating the status of the submitted patches
  - Letting know if the patches have been received, reviewed, applied or rejected.
  - Expected to be done in real time.

Expectation of the upper maintainer

- Don't introduce any breakage.
- No merge conflict.
- No regression.

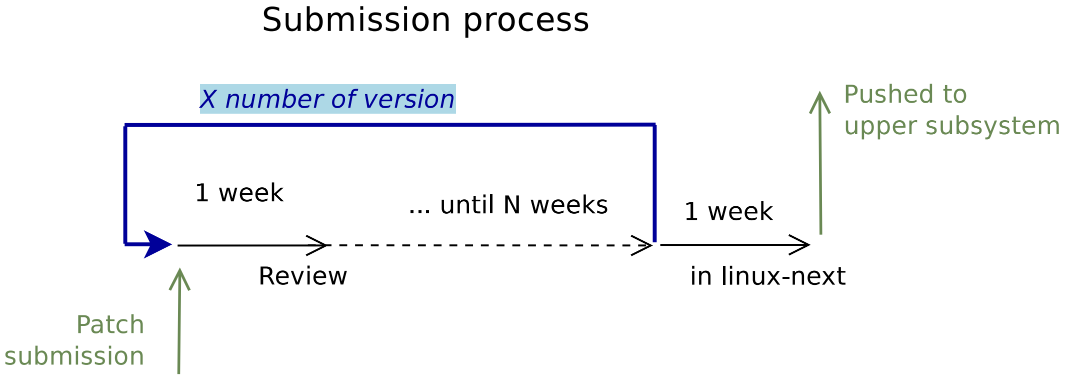
(Image credit: Mike Pennington under license CC BY-SA 2.0)

Timeline for the submission of a patch 1/2

- At least one week between submission and being applied
  - Let time to interested to review the series
  - Could be shorter for a new version of a series already reviewed
- Stay in linux-next one week before being submitted to the upper subsystem
  - Allow to fix merge conflicts before creating an immutable branch.
  - Could be shorter if already been in linux-next before or if the change is well contained.

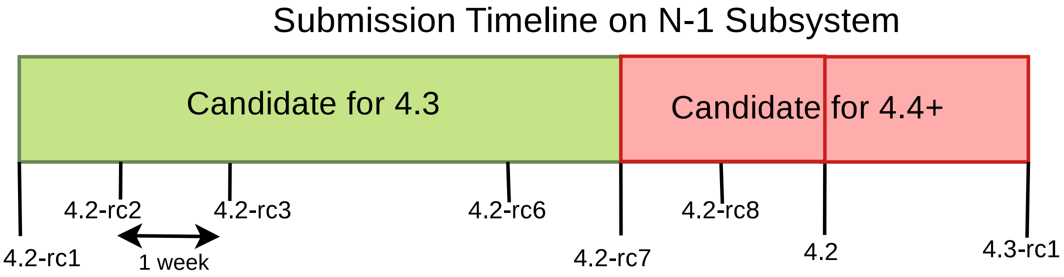
Timeline for the submission of a patch 2/3

- Deeper is the subsystem, longer will be the time between submission and merged in mainline



Timeline for the submission of a patch 3/3

- As the Linux release candidate are weekly, then for a subsystem at N-1, series submitted after -rc6 (or rc7) won't be in next release.



Accepting a patch

- Obvious criteria
  - Must respect the coding rules (use checkpatch for this).
  - Must compiled without warning.
- No regression.
- Testing the hardware is nice to have but not mandatory.
  - For a new device feature or device you can assume it was tested by the submitter.
  - Ask for a tested by from other user if you have any doubt.
  - Rely on testing farm if you can.
- Be careful of dependencies to the other subsystem.

Organization of the subsystem git tree

- At least 2 branches:
  - `current` for gathering the fixes of the current release candidate.
  - `for-next` for gathering the patches for the next release candidate.
- Could be useful to have a third branch for the release candidate after.
- Could have topic branches:
  - For big subsystem such as `arm-soc`.
  - To let other subsystems merge your subsystem related part of series (see later).
- Based on the -rc1 to make the merge easier.

The stable kernels

- Most of the user use kernel from a distribution.
- Most of the distribution use stable kernel
- When receiving a fix always ask if it could be useful for older kernel.
- Tag the commit with `Cc:<stable@vger.kernel.org>`.
- Even better use the tag `Fixes: SHA-1_ID ("title of the patch")`.

The linux-next kernel 1/2

- The place where all are merged the commits expected to be in kernel after the next merge window closes.
- How to use it as a maintainer
  - The branches merged in linux-next have to be declared to Stephen Rothwell.
  - Send him an email with the name of the repository and the branch to merge.
  - Do not have to be an immutable branch: all the branches are merged again for each linux-next release (on a daily basis).

The linux-next kernel 2/2

- Benefit of being in `linux-next`
  - Being merged every day with all the other branches allows detecting the merge conflict early.
  - As a bonus Stephen often came with the resolution of the conflict.
  - Used by the autobuilder such as `0-Day` done by `01.org` from `Intel` or the `kernelci` supported by `Linaro`.

Dealing with your own patches

- You are a maintainer but you remain a developer.
- You have the possibility to directly applied your own patches.
- Not really in the spirit of an open development.
- Still good to have review and suggestion.
- However most of the time you won't get a review as you are supposed to be the one who review!
- But still apply the submission process: waiting at least one week after submitting on the mailing list before applying it in your `next` branch.

Coordinating with the co-maintainers 1/2

- Subsystems maintained more and more often by several's peoples.
- Benefits:
  - Allow to be more responsive especially if located in distant timezone.
  - Avoid having a stalled subsystem during holidays.
  - Ease the turn over: easier to leave and easier to join a team.
- Drawbacks
  - Need to able to find an agreement in case of opposite opinions.
  - Need to coordinate.

Coordinating with the co-maintainers 2/2

- Each co-maintainer has her/his own interests and fields of expertise.
  - Spread the review.
  - Allow staying focused.
- An acked-by given by a co-maintainer is enough.
- Only one co-maintainer gathering the patches and taking care of the pull requests for a given kernel release cycle.
  - Easier to keep the track of the submitted patches.
  - The git repository remains shared at least for emergency.
  - Better to decide in advance who will be the next in charge.
- Coordinating by email is fine most of the time.

Coordinating with the maintainers of other subsystem 1/2

- Some series modify several subsystems in the same time.
- Dependencies between the patches.
- We want that the kernel be *bisectable*.
- The order in which the patches are applied matters.
- Can't predict in which order the subsystem will be merged.
- Need to synchronize with the maintainers of other subsystems to solve this.

Coordinating with the maintainers of other subsystem 2/2

- One maintainer takes all the series:
  - Will have commits modifying another subsystem in her/his git tree.
  - May cause conflict merge.
- One maintainer create an immutable branch
  - A topic branch with only the patch from the series.
  - Will be in both tree: it will avoid the merge conflict.
  - If a fix is needed it can't be squashed, have to be a separate commit.
- Merging the series in two kernel releases:
  - No merge conflict.
  - No immutable branch.
  - But the feature is delayed of at least 3 months.
  - Still possible to have the feature by delaying the clean-up in the second release.

Submitting the gathered patches 1/2

▶ Identify the patch to apply when reading the emails.	`M-x gnus-registry-set-article-mark` under `emacs` or by using `patchwork`.
▶ Apply them on your branch.	`M-x dvc-gnus-article-apply-patch` under `emacs`.
▶ Add your Signed-off-by (as you are going to submit them you have to do it).	`git commit --amend -s --no-edit`

Submitting the gathered patches 2/2

▶ Signed your branch	`git tag -s tag_name branch_name`
▶ Push your branch on your public repository	`git push public_repo tags/tag_name`
▶ Generate the pull request cover letter:	`git request-pull previous_tag public_repo \ tags/tag_name | cat` `previous_tag` is either the tag previously pulled during the last request or the `rc1` of the current kernel.

Final words

- Find the good balance between maintainer duty and submitter expectation.
- Be nice and helpful with the submitter especially the new ones.
- Remain vigilant about the code quality and stability of the kernel.

Questions?

Gregory CLEMENT
gregory.clement@free-electrons.com