how and where to send patch

Abdul Rahim

23 July 2024

- the `get_maintainer.pl` tells you whome to send the patch to
- You should send the patch to *maintainers, commit signers, supporters, and all the mailing lists* shown in the `get\_maintainer.pl`'s output.
- Mailing lists are on the "cc" and the rest are on the "To" list when a patch is sent

```
scripts/get_maintainer.pl drivers/media/usb/
    uvc/uvc_driver.c
Laurent Pinchart <laurent.
    pinchart@ideasonboard.com> (maintainer:USB
    VIDEO CLASS)
Mauro Carvalho Chehab <mchehab@kernel.org> (
    maintainer:MEDIA INPUT INFRASTRUCTURE (V4L/
    DVB))
linux-media@vger.kernel.org (open list:USB
    VIDEO CLASS)
linux-kernel@vger.kernel.org (open list)
```

- Then you can run:

```
git format-patch -1 <commit ID> --to=laurent.pinchart@ideasonboard.com --to=mchehab@kernel.org --cc=linux-media@vger.kernel.org --cc=linux-kernel@vger.kernel.org
```

- This will generate a patch that you can send using:

```
git send-email <patch_file>
```

patch review

- Your patch will get comments from reviewers with suggestions. wait for a minimum of one week before requesting a response(might take longer).
- When you send a new version of your patch, add version history describing the changes made in the new version. The **right place for the version history is after the “—” below the Signed-off-by tag and the start of the changed file list**, as shown in the *screenshot*.
- Everything between the Signed-off-by and the diff is just for the reviewers, and will not be included in the commit. Please don’t include version history in the commit log.

Best Practices for Sending Patches

- Document your change and include relevant testing details and results of that testing.
- Signed-off-by should be the last tag.
- Copy mailing lists and maintainers/developers suggested by `scripts/get_maintainer.pl`
- Always thank the reviewers for their feedback and address them.

- When working on a patch based on a suggested idea, make sure to give credit using the Suggested-by tag. Other tags used for giving credit are Tested-by, Reported-by.
- In general, getting response and comments is a good sign that the community likes the patch and wants to see it improved. Silence is what you want to be concerned about. If you don’t hear any response back from the maintainer after a week, feel free to either send the patch again, or send a gentle “ping” - something like “Hi, I know you are busy, but have you found time to look at my patch?”
- Expect to receive comments and feedback at any time during the review process.

- Stay engaged and be ready to fix problems, if any, **after the patch gets accepted into linux-next for integration into the mainline**.
- Kernel build and Continuous Integration (CI) bots and rings usually find problems.
- When a patch gets accepted, you will either see an email from the maintainer or an automated patch **accepted email** with information on **which tree it has been applied to**, and some estimate on when you can expect to see it in the mainline kernel.

- Not all maintainers might send an email when the patch gets merged. The patch could stay in linux-next for integration until the next merge window, before it gets into Linus’s tree. Unless the patch is an **actual fix to a bug in Linus’s tree**, in which case, it may go directly into his tree.

Sometimes you need to send multiple related patches. This is useful for grouping, say, to **group driver clean up patches for one particular driver into a set**, or grouping patches that are part of a **new feature** into one set.

```
git format-patch -2 -s --cover-letter --thread
    --subject-prefix="PATCH v3" --to= "name"
    --cc="name"
```

will create a threaded patch series that includes the top two commits and generated cover letter template. It is a good practice to send a cover letter when sending a patch series.

- Including patch series version history in the cover letter will help reviewers get a quick snapshot of changes from version to version. When a maintainer accepts a patch, the maintainer assumes maintenance responsibility for that patch. As a result, maintainers have decision power on how to manage patch flow into their individual sub-system(s) and they also have individual preferences. Be prepared for maintainer-to-maintainer differences in commit log content and sub-system specific coding styles.
- *sample*

git hooks

- Git includes some hooks for **scripts** that can be run **before or after specific git commands are executed**.
- Checking the patch for compliance and errors can be automated using **git pre-commit** and **post-commit hooks**. The post-commit hook is run after you make a git commit with the git commit command.

- If you don’t already have `/usr/share/codespell/dictionary.txt`, do:

```
sudo apt-get install codespell
```

- If you already have a `.git/hooks/post-commit` file, move it to `.git/hooks/post-commit.sample`. git will not execute files with the .sample extension.
- Then, edit the `.git/hooks/post-commit` file to contain only the following two lines:

```
#!bash
#!/bin/sh
exec git show --format=email HEAD | ./scripts/
    checkpatch.pl --strict --codespell
# Make sure the file is executable:
chmod a+x .git/hooks/post-commit
```

- After you make the commit, this hook will output any checkpatch errors or warnings that your patch creates
- if you want to avoid running this hook use:

```
git add .
git commit --amend
```