Creating Pull Requests

This chapter describes how maintainers can create and submit pull requests to other maintainers. This is useful for transferring changes from one maintainers tree to another maintainers tree.

This document was written by Tobin C. Harding (who at that time, was not an experienced maintainer) primarily from comments made by Greg Kroah-Hartman and Linus Torvalds on LKML. Suggestions and fixes by Jonathan Corbet and Mauro Carvalho Chehab. Misrepresentation was unintentional but inevitable, please direct abuse to Tobin C. Harding <me@tobin.cc>.

Original email thread:

```
https://lore.kernel.org/r/20171114110500.GA21175@kroah.com
```

Create Branch

To start with you will need to have all the changes you wish to include in the pull request on a separate branch. Typically you will base this branch off of a branch in the developers tree whom you intend to send the pull request to.

In order to create the pull request you must first tag the branch that you have just created. It is recommended that you choose a meaningful tag name, in a way that you and others can understand, even after some time. A good practice is to include in the name an indicator of the subsystem of origin and the target kernel version.

Greg offers the following. A pull request with miscellaneous stuff for drivers/char, to be applied at the Kernel version 4.15-rc1 could be named as `char-misc-4.15-rc1`. If such tag would be produced from a branch named `char-misc-next`, you would be using the following command:

```
git tag -s char-misc-4.15-rc1 char-misc-next
```

that will create a signed tag called `char-misc-4.15-rc1` based on the last commit in the `char-misc-next` branch, and sign it with your gpg key (see :ref:`Documentation/maintainer/configure-git.rst <configuregit>`).

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\maintainer\[linux-master][Documentation][maintainer]pull-requests.rst, line 42); *backlink***
>
> Unknown interpreted text role "ref".

Linus will only accept pull requests based on a signed tag. Other maintainers may differ.

When you run the above command `git` will drop you into an editor and ask you to describe the tag. In this case, you are describing a pull request, so outline what is contained here, why it should be merged, and what, if any, testing has been done. All of this information will end up in the tag itself, and then in the merge commit that the maintainer makes if/when they merge the pull request. So write it up well, as it will be in the kernel tree for forever.

As said by Linus:

```
Anyway, at least to me, the important part is the *message*. I want
to understand what I'm pulling, and why I should pull it. I also
want to use that message as the message for the merge, so it should
not just make sense to me, but make sense as a historical record
too.

Note that if there is something odd about the pull request, that
should very much be in the explanation. If you're touching files
that you don't maintain, explain _why_. I will see it in the
diffstat anyway, and if you didn't mention it, I'll just be extra
suspicious.  And when you send me new stuff after the merge window
(or even bug-fixes, but ones that look scary), explain not just
what they do and why they do it, but explain the _timing_. What
happened that this didn't go through the merge window..

I will take both what you write in the email pull request _and_ in
the signed tag, so depending on your workflow, you can either
describe your work in the signed tag (which will also automatically
make it into the pull request email), or you can make the signed
tag just a placeholder with nothing interesting in it, and describe
the work later when you actually send me the pull request.

And yes, I will edit the message. Partly because I tend to do just
trivial formatting (the whole indentation and quoting etc), but
partly because part of the message may make sense for me at pull
time (describing the conflicts and your personal issues for sending
it right now), but may not make sense in the context of a merge
commit message, so I will try to make it all make sense. I will
also fix any speeling mistaeks and bad grammar I notice,
particularly for non-native speakers (but also for native ones
;^). But I may miss some, or even add some.

                Linus
```

Greg gives, as an example pull request:

```
Char/Misc patches for 4.15-rc1

Here is the big char/misc patch set for the 4.15-rc1 merge window.
Contained in here is the normal set of new functions added to all
of these crazy drivers, as well as the following brand new
subsystems:
        - time_travel_controller: Finally a set of drivers for the
          latest time travel bus architecture that provides i/o to
          the CPU before it asked for it, allowing uninterrupted
          processing
        - relativity_shifters: due to the affect that the
          time_travel_controllers have on the overall system, there
          was a need for a new set of relativity shifter drivers to
          accommodate the newly formed black holes that would
          threaten to suck CPUs into them.  This subsystem handles
          this in a way to successfully neutralize the problems.
          There is a Kconfig option to force these to be enabled
          when needed, so problems should not occur.

All of these patches have been successfully tested in the latest
linux-next releases, and the original problems that it found have
all been resolved (apologies to anyone living near Canberra for the
lack of the Kconfig options in the earlier versions of the
linux-next tree creations.)

Signed-off-by: Your-name-here <your_email@domain>
```

The tag message format is just like a git commit id. One line at the top for a "summary subject" and be sure to sign-off at the bottom.

Now that you have a local signed tag, you need to push it up to where it can be retrieved:

```
git push origin char-misc-4.15-rc1
```

Create Pull Request

The last thing to do is create the pull request message. `git` handily will do this for you with the `git request-pull` command, but it needs a bit of help determining what you want to pull, and on what to base the pull against (to show the correct changes to be pulled and the diffstat). The following command(s) will generate a pull request:

```
git request-pull master git://git.kernel.org/pub/scm/linux/kernel/git/gregkh/char-misc.git/ char-misc-4.15-
```

Quoting Greg:

```
This is asking git to compare the difference from the
'char-misc-4.15-rc1' tag location, to the head of the 'master'
branch (which in my case points to the last location in Linus's
tree that I diverged from, usually a -rc release) and to use the
git:// protocol to pull from.  If you wish to use https://, that
can be used here instead as well (but note that some people behind
firewalls will have problems with https git pulls).

If the char-misc-4.15-rc1 tag is not present in the repo that I am
asking to be pulled from, git will complain saying it is not there,
a handy way to remember to actually push it to a public location.

The output of 'git request-pull' will contain the location of the
git tree and specific tag to pull from, and the full text
description of that tag (which is why you need to provide good
information in that tag).  It will also create a diffstat of the
pull request, and a shortlog of the individual commits that the
pull request will provide.
```

Linus responded that he tends to prefer the `git://` protocol. Other maintainers may have different preferences. Also, note that if you are creating pull requests without a signed tag then `https://` may be a better choice. Please see the original thread for the full discussion.

Submit Pull Request

A pull request is submitted in the same way as an ordinary patch. Send as inline email to the maintainer and CC LKML and any sub-system specific lists if required. Pull requests to Linus typically have a subject line something like:

```
[GIT PULL] <subsystem> changes for v4.15-rc1
```