How to Get Your Change Into the Linux Kernel

Based on the Linux kernel file Documentation/SubmittingPatches

For a person or company who wishes to submit a change to the Linux kernel, the process can sometimes be daunting if you're not familiar with "the system." This text is a collection of suggestions which can greatly increase the chances of your change being accepted.

If you are submitting a driver, also read Documentation/SubmittingDrivers.

"diff -up"

Use "diff -up" or "diff -uprN" to create patches.

All changes to the Linux kernel occur in the form of patches, as generated by diff(1). When creating your patch, make sure to create it in "unified diff" format, as supplied by the '-u' argument to diff(1). Also, please use the '-p' argument which shows which C function each change is in - that makes the resultant diff a lot easier to read. Patches should be based in the root kernel source directory, not in any lower subdirectory.

To create a patch for a single file, it is often sufficient to do:

```
SRCTREE= linux-2.4
MYFILE=  drivers/net/mydriver.c
cd $SRCTREE
cp $MYFILE $MYFILE.orig
vi $MYFILE # make your change
cd ..
diff -up $SRCTREE/$MYFILE{.orig,} > /tmp/patch
```

To create a patch for multiple files, you should unpack a "vanilla", or unmodified kernel source tree, and generate a diff against your own source tree. For example:

```
MYSRC= /devel/linux-2.4
tar xvfz linux-2.4.0-test11.tar.gz
mv linux linux-vanilla
wget http://www.moses.uklinux.net/patches/dontdiff
diff -uprN -X dontdiff linux-vanilla $MYSRC > /tmp/patch
rm -f dontdiff
```

"dontdiff" is a list of files which are generated by the kernel during the build process, and should be ignored in any diff(1)-generated patch.

Make sure your patch does not include any extra files which do not belong in a patch submission. Make sure to review your patch -after- generated it with diff(1), to ensure accuracy.

If your changes produce a lot of deltas, you may want to look into splitting them into individual patches which modify things in logical stages, this will facilitate easier reviewing by other kernel developers, very important if you want your patch accepted.

Describe your changes.

Describe the technical detail of the change(s) your patch includes and be as specific as possible. The WORST descriptions possible include things like "update driver X", "bug fix for driver X", or "this patch includes updates for subsystem X. Please apply."

If your description starts to get long, that's a sign that you probably need to split up your patch. See the next section.

Separate your changes.

Separate each logical change into its own patch.

For example, if your changes include both bug fixes and performance enhancements for a single driver, separate those changes into two or more patches. If your changes include an API update, and a new driver which uses that new API, separate those into two patches.

On the other hand, if you make a single change to numerous files, group those changes into a single patch. Thus a single logical change is contained within a single patch.

If one patch depends on another patch in order for a change to be complete, that is OK. Simply note "this patch depends on patch X" in your patch description.

Select e-mail destination.

Look through the MAINTAINERS file and the source code, and determine if your change applies to a specific subsystem of the kernel, with an assigned maintainer. If so, e-mail that person.

If no maintainer is listed, or the maintainer does not respond, send your patch to the primary Linux kernel developer's mailing list, linux-kernel@vger.kernel.org. Most kernel developers monitor this e-mail list, and can comment on your changes.

Linus Torvalds is the final arbiter of all changes accepted into the Linux kernel. His e-mail address is <torvalds@osdl.org>. He gets a lot of e-mail, so typically you should do your best to -avoid- sending him e-mail.

Patches which are bug fixes, are "obvious" changes, or similarly require little discussion should be sent or CC'd to Linus. Patches which require discussion or do not have a clear advantage should usually be sent first to linux-kernel. Only after the patch is discussed should the patch then be submitted to Linus.

For small patches you may want to CC the Trivial Patch Monkey trivial@rustcorp.com.au set up by Rusty Russell; which collects "trivial" patches. Trivial patches must qualify for one of the following rules:

- Spelling fixes
- Warning fixes (cluttering with useless warnings is bad)
- Compilation fixes (only if they are actually correct)
- Runtime fixes (only if they actually fix things)
- Removing use of deprecated functions/macros (eg. check_region).
- Contact detail and documentation fixes
- Non-portable code replaced by portable code (even in arch-specific, since people copy, as long as it's trivial)
- Any fix by the author/maintainer of the file. (ie. patch monkey in re-transmission mode)

Select your CC (e-mail carbon copy) list.

Unless you have a reason NOT to do so, CC linux-kernel@vger.kernel.org.

Other kernel developers besides Linus need to be aware of your change, so that they may comment on it and offer code review and suggestions. linux-kernel is the primary Linux kernel developer mailing list. Other mailing lists are available for specific subsystems, such as USB, framebuffer devices, the VFS, the SCSI subsystem, etc. See the MAINTAINERS file for a mailing list that relates specifically to your change.

Even if the maintainer did not respond in the previous step, make sure to ALWAYS copy

the maintainer when you change their code.

No MIME, no links, no compression, no attachments. Just plain text.

Linus and other kernel developers need to be able to read and comment on the changes you are submitting. It is important for a kernel developer to be able to "quote" your changes, using standard e-mail tools, so that they may comment on specific portions of your code.

For this reason, all patches should be submitting e-mail "inline". WARNING: Be wary of your editor's word-wrap corrupting your patch, if you choose to cut-n-paste your patch.

Do not attach the patch as a MIME attachment, compressed or not. Many popular e-mail applications will not always transmit a MIME attachment as plain text, making it impossible to comment on your code.

Exception: If your mailer is mangling patches then someone may ask you to re-send them using MIME.

E-mail size

When sending patches to Linus, always follow the previous step.

Large changes are not appropriate for mailing lists, and some maintainers. If your patch, uncompressed, exceeds 40 kB in size, it is preferred that you store your patch on an Internet-accessible server, and provide instead a URL (link) pointing to your patch.

Name your kernel version.

It is important to note, either in the subject line or in the patch description, the kernel version to which this patch applies. If the patch does not apply cleanly to the latest kernel version, Linus will not apply it.

Don't get discouraged. Re-submit.

After you have submitted your change, be patient and wait. If Linus likes your change and applies it, it will appear in the next version of the kernel that he releases.

However, if your change doesn't appear in the next version of the kernel, there could be any number of reasons. It's YOUR job to narrow down those reasons, correct what was wrong, and submit your updated change.

It is quite common for Linus to "drop" your patch without comment. That's the nature of the system. If he drops your patch, it could be due to

- Your patch did not apply cleanly to the latest kernel version
- Your patch was not sufficiently discussed on linux-kernel.
- A style issue (see previous sections),
- An e-mail formatting issue (re-read this section)
- A technical problem with your change
- He gets tons of e-mail, and yours got lost in the shuffle
- You are being annoying (See Figure 1)

When in doubt, solicit comments on linux-kernel mailing list.

Include PATCH in the subject

Due to high e-mail traffic to Linus, and to linux-kernel, it is common convention to prefix your subject line with [PATCH]. This lets Linus and other kernel developers more easily distinguish patches from other e-mail discussions.

Sign your work

To improve tracking of who did what, especially with patches that can percolate to their final resting place in the kernel through several layers of maintainers, we've introduced a "sign-off" procedure on patches that are being emailed around.

The sign-off is a simple line at the end of the explanation for the patch, which certifies that you wrote it or otherwise have the right to pass it on as a open-source patch. The rules are pretty simple: if you can certify the below:

> Developer's Certificate of Origin 1.0
>
> By making a contribution to this project, I certify that:
>
> (a) The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or
>
> (b) The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or
>
> (c) The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.

then you just add a line saying

> Signed-off-by: Random J Developer <random@developer.org>

Some people also put extra tags at the end. They'll just be ignored for now, but you can do this to mark internal company procedures or just point out some special detail about the sign-off.