

How to make a GNU/Linux kernel patch?

Robert Berger

<http://www.ReliableEmbeddedSystems.com>
robert.berger@ReliableEmbeddedSystems.com

Intro

Did you hear about the flamewar back in 1992 between Andy Tannenbaum [1] and Linus Torvalds entitled "Linux is obsolete"? [2] The famous Professor Andy Tannenbaum told the 20-something year old - and at this time still unknown - Linus: "Be thankful you are not my student. You would not get a high grade for such a design :-)". The rest is history,...

You might never have heard about the brainchild of Andy called Minix, but probably you've heard about an operating system called GNU/Linux. It might be already running on your mobile phone, your set-top box, in one of the many embedded systems in your car, in some other appliances of yours or the machines hosting web sites and routing packets over the internet.

Notorious GNU/Linux users/supporters



Figure 1: Linus Torvalds-Linuxfoundation



Figure 2: Sergey Mikhailovich Brin-Google

Linus [3] is the creator of GNU/Linux and remains the ultimate authority on what new code is incorporated into the standard GNU/Linux kernel. Sergey [4] is the co-founder of Google where the biggest GNU/Linux cluster on this planet [5] is used to satisfy your search requests and some more.

Have a close look at those pictures. Would you really entrust your business in technology created/used by those guys?

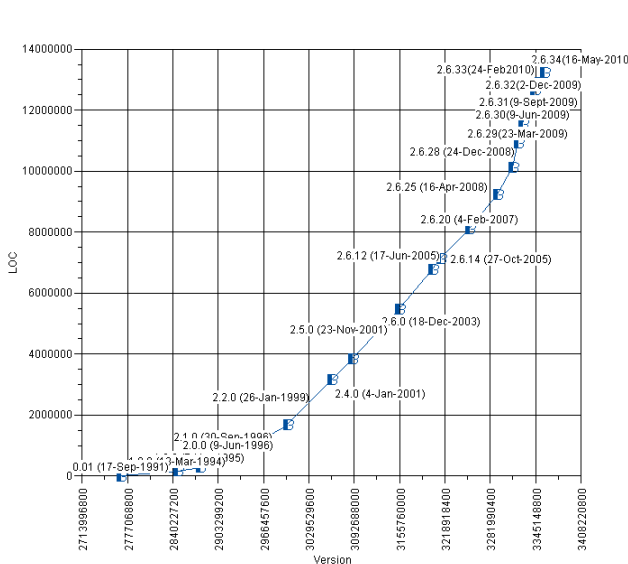
Who is developing GNU/Linux?

company	contribution	company	contribution
Nobody	16.9%	Texas Instruments	1.4%
Red Hat	11.8%	Analog Devices	1.4%
Intel	8.0%	academics	1.4%
unknown	7.0%	Atheros Comm.	1.3%
IBM	5.2%	HP	1.1%
Novell	4.9%	Pengutronix	1.1%
consultants	2.4%	Wolfson Micro	1.0%
Renesas Tech	2.2%	Broadcom	0.8%
Nokia	2.2%	Marvell	0.8%
Oracle	1.9%	NTT	0.7%
Fujitsu	1.8%	NetApp	0.7%
AMD	1.6%	MontaVista	0.7%

Table 1: kernel version 2.6.35-rc1

Who are the geeks developing the GNU/Linux kernel? One urban myth is that it's developed by hackers in their free time, but from the table above you can see that the majority of code is contributed by people who are paid for doing so and only 16.9% of the code comes from people who claim not to be working for any company. This means that the GNU/Linux kernel might be made by geeks - and possibly 16.9% of them are hacking in their spare time for free in their basements - but most of them are professionals and make a living from writing (usually high quality) code.

Defying the laws of Cocomo



estimated as €1,025,553,430 [10] [11] and about 985 developers would be needed over a span of just under 14 years.

What's the mainline kernel? [12]

It's the base for GNU/Linux distributions and maintained by Linus Torvalds who is the one and only who can commit patches to it. Obviously it's too big for a single person to comprehend so there is a lieutenant system of subsystem maintainers build around a "chain of trust". Those trusted people send patches to Linus who adds them to mainline. When you want to write your first kernel patch you are most likely not one of those "trusted" people and need to send it to a subsystem tree [13], linux-next [12] or the staging tree. The staging tree lives in mainline under drivers/staging and there is plenty of work which needs to be done. The only requirement for a patch is to comply with various "style" issues we'll discuss later and to compile. So it's a great place to start with.

Why would you want to bring your code to mainline?

The philosophical answer is: "To be part of the community." The commercial answer is: "Maintenance costs are much cheaper that way and mainline code is most likely going to be of higher quality due to a review process." You should post your initial ideas early on the relevant mailing lists to avoid multiple solutions to the same problem and to start it the right way. After your ideas get blessed and there are no more design obstacles post your implementation early and don't wait until you think it's perfect. From two equally fit solutions the first one will be in mainline. Of course it may always be replaced by a better solution later on. The closer your kernel is to mainline the smaller your maintenance costs will be since the community will help with the maintenance work.

Once upon a time

The original process was to download the kernel - a tarball, which fit on a floppy disk and email patches back to Linus. He would read every patch and comment on every email. The updated process was to download the kernel and email the patches to subsystem maintainers. Those would collect and check the patches and send them to Linus. This model also did not scale very well since Linus had kids (and less time) and the project grew in size. One central co-ordinator with a traditional centralized version control system did not work very well and changelogs were hand edited, incomplete and hard to manage. Bitkeeper was used, but this was not Open Source so it was reverse engineered [14] and eventually dropped.

How is it done today?

Clearly something better was needed so Linus wrote a decentralized/distributed version control system called git. Linus Torvalds about "git", which is British English slang for a stupid or unpleasant person: [15] "I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git." [16]

Learn git basics in less than 5 minutes

1 `man gittutorial`

Get the latest and greatest DENX powerpc kernel tree

```
1 cd $ELDK_PREFIX/eldk-4.2-ppc_4xx/usr/src
2 git clone git://git.denx.de/linux-2.6-denix.git linux-2.6-denix-latest
3 git status
4 git checkout master
5 git checkout -b kernel-patch
6 git branch
```

GNU/Linux Kernel Source Tree

Documentation is a self explanatory folder in the kernel tree. It contains text files describing various aspects of the kernel, problems and "gotchas"... There are also several subdirectories for topics which require more extensive explanations.

Documentation/HOWTO contains the following:

HOWTO do Linux kernel development

This is the be-all, end-all document on this topic. It contains instructions on how to become a Linux kernel developer and how to learn to work with the Linux kernel development community. It tries to not contain anything related to the technical aspects of kernel programming, but will help point you in the right direction for that.

If anything in this document becomes out of date, please send in patches to the maintainer of this file, who is listed at the bottom of the document.

GNU/Linux Kernel coding style

Documentation/CodingStyle starts like this: *"First off, I'd suggest printing out a copy of the GNU coding standards, and NOT read it. Burn them, it's a great symbolic gesture."* The reason it's highly recommended for every reasonably sized non trivial software project to have a common coding style is that the brain detects patterns. Consistency is very important especially when many people are reading your code. A nice side effect is that if you write code under the assumption that many people will read it you will most likely try to make it look good.

Apropos spelunking:

- I want to read your code
- I want you to read my code
- I want you to fix my bugs
- I want you to change my code
- I want to build on your code

It's always intriguing to me that in order to become a composer you need to read music sheets and listen to music, to become a writer you need to read books from famous authors, to become an architect you study buildings of others. It's just many programmers who think that their code is the best in the universe and don't bother to read other people's code. Please note, that every line of GNU/Linux kernel code is reviewed by at least two people (and git can show you who those people are).

What follows are some examples of the GNU/Linux kernel coding style:

CodingStyle/Indentation tabs 8 chars

```
1 switch (suffix) {
2     case 'G':
3         mem <<= 30;
4         break;
5     case 'M':
6         mem <<= 20;
7         break;
8     case 'K':
9         mem <<= 10;
10        /* fall through */
11    default:
12        break;
13 }
```

CodingStyle/80 chars max.

```
1 void fun(int a, int b, int c)
2 {
3     if (condition)
4         printk(KERN_WARNING "long_printk_with_"
5                 "param_a:_%u_b:_%u_"
6                 "c:_%u\n", a, b, c);
7     else
8         next_statement;
9 }
```

CodingStyle/braces

```
1 /* don't use braces with single statement */
2 if (condition)
3     action();
4 /* This does not apply if one branch of a conditional statement is a single
5 statement. Use braces in both branches.*/
6 if (condition) {
7     do_this();
8     do_that();
9 } else {
10     otherwise();
11 }
```

CodingStyle/centralized exit

```
1 int fun(int a)
2 {
3     ...
4     if (condition1) {
5         while (loop1) {
6             ...
7         }
8         goto out;
9     }
10    ...
11 out: kfree(buffer);
12     return result;
13 }
```

Create your first kernel patch

Let's print something at the kernel main init

```
1 diff --git a/init/main.c b/init/main.c
2 index f3d7cf1..af7b957 100644
3 --- a/init/main.c
4 +++ b/init/main.c
5 @@ -547,6 +547,8 @@ asmlinkage void __init start_kernel(void)
6     char * command_line;
7     extern struct kernel_param __start__param[], __stop__param[];
8
9 + printk("The kernel is starting up!\n");
10 +
11     smp_setup_processor_id();
```

Build and install the kernel

```
1 source ~/projects/ldd/ldd_for_trainer/env.sh
2 cd $ELDK_PREFIX/eldk-4.2-ppc_4xx/usr/src/linux-2.6-denix-latest
3 make ARCH=powerpc CROSS_COMPILE=ppc_4xx- mrproper
4 make ARCH=powerpc CROSS_COMPILE=ppc_4xx- 40x/kilauea_defconfig
5 make ARCH=powerpc CROSS_COMPILE=ppc_4xx- menuconfig
6 make ARCH=powerpc CROSS_COMPILE=ppc_4xx- uImage EXTRAVERSION=-classic-rber
7 make ARCH=powerpc CROSS_COMPILE=ppc_4xx- modules EXTRAVERSION=-classic-rber
8 make ARCH=powerpc CROSS_COMPILE=ppc_4xx-
9     INSTALL_MOD_PATH=$ELDK_PREFIX/eldk-4.2-ppc_4xx/ppc_4xx modules_install
10 make ARCH=powerpc CROSS_COMPILE=ppc_4xx- kilauea.dtb
11 cp
12     $ELDK_PREFIX/eldk-4.2-ppc_4xx/usr/src/linux-2.6-denix-latest/arch/powerpc/boot/kilauea.dtb
13     /tftpboot/kilauea/kilauea.dtb
14 cp $ELDK_PREFIX/eldk-4.2-ppc_4xx/usr/src/linux-2.6-denix-latest/arch/powerpc/boot/uImage
15     /tftpboot/kilauea/uImage
```

We assume that our target is a Kilauea board [17], our development kit is the ELDK [18], the root file system (rootfs) is exported via NFS and that our boot loader is u-boot [19] which loads the kernel and the flat device tree via tftp. At this point we should have all files needed and be ready to roll.

Does the modification do as expected?

Assuming the kernel boots one way to find out what it did is to have a look at the kernel message buffer, which can be done with `dmesg`.

```
1 dmesg | more
2
3 ...
4 The kernel is starting up!
5 Linux version 2.6.36-classic-rber-dirty (rber@eldk-ppc) (gcc version 4.2.2) #1 Thu Sep 2
   00:57:41 EEST 2010
6 ...
```

As you can see the message: *"The kernel is starting up!"* is printed, which means that the change seems to work.

Let's create a patch

`git` helps us creating the patch:

```
1 cd $ELDK_PREFIX/eldk-4.2-ppc_4xx/usr/src/linux-2.6-denx-latest
2 git status
3 git diff
4 git commit init/main.c -m "my_first_kernel_patch"
5 git format-patch master..kernel-patch -o ../patches
```

Note that in real life you should write in the changelog something more meaningful than *"my first kernel patch"*.

Are we ready to roll?

A script called `checkpatch.sh` checks for various "style" issues:

```
1 ./scripts/checkpatch.pl ../patches/0001-my-first-kernel-patch.patch
```

That's what the script complains about (WARNINGS and ERRORS):

```
1 WARNING: printk() should include KERN_ facility level
2 #18: FILE: init/main.c:550:
3 + printk("The_kernel_is_starting_up\n!");
4
5 ERROR: Missing Signed-off-by: line(s)
6 total: 1 errors, 1 warnings, 8 lines checked
7
8 ../patches/0001-my-first-kernel-patch.patch has style problems, please review. If any of
   these errors are false positives report them to the maintainer, see CHECKPATCH in
   MAINTAINERS.
```

What's this maintainers stuff about?

The output from the previous script mentioned something about maintainers. Every file in the kernel should have a maintainer and those are supposed to be hand written into the `MAINTAINERS` file, but a script in combination with `git` provides a more automated way to find out who are the maintainers. Let's see who the maintainers for the line we changed in our patch are:

```
1 cd $ELDK_PREFIX/eldk-4.2-ppc_4xx/usr/src/linux-2.6-denx-latest
2 ./scripts/get_maintainer.pl ../patches/0001-my-first-kernel-patch.patch
3 Pekka Enberg <penberg@cs.helsinki.fi>
4 Ingo Molnar <mingo@elte.hu>
5 Benjamin Herrenschmidt <benh@kernel.crashing.org>
6 linux-kernel@vger.kernel.org
```

When you send your patch to the community later on those people should certainly be included in the mail.

Let's fix the WARNING:

```
1 vim init/main.c +550
2
3     char * command_line;
4     extern struct kernel_param __start__param[], __stop__param[];
5
6 - printk("The kernel is starting up\n!");
7 + printk(KERN_INFO "The kernel is starting up\n!");
8
9     smp_setup_processor_id();
10
11 git commit init/main.c -m "fixed checkpatch WARNING - added KERN_INFO"
```

Let's fix the ERROR: This can be done by adding

[format]

```
signoff = true
```

to your .gitconfig as a permanent solution or by passing -s to the git command line.

```
1 this fixes the checkpatch ERROR (see .gitconfig):
2 git format-patch master..kernel-patch -o ../patches -s
3
4 cat ../patches/0002-fixed-checkpatch-WARNING.patch
5 +Signed-off-by: Robert Berger <git.rber@gmail.com>
6
7 - printk("The kernel is starting up\n!");
8 + printk(KERN_INFO "The kernel is starting up\n!");
9
10     smp_setup_processor_id();
```

What's this Signed-off-by stuff about?

Have a look at Documentation/SubmittingPatches:

```
1 By making a contribution to this project, I certify that:
2 (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
3 (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
4 By making a contribution to this project, I certify that:
5 (c) The contribution was provided directly to me by some other person who certified (a),
   (b) or (c) and I have not modified it.
```

Create a new patch

```
1 cd $ELDK_PREFIX/eldk-4.2-ppc_4xx/usr/src/linux-2.6-denx-latest
2 git checkout master
3 git checkout -b kernel-patch-staging
4 git log kernel-patch
5 git cherry-pick -n 5bd6228c3019dc805dffbc69b82fce4240bbc63
6 git cherry-pick -n dc87063a24af0bc764bcbe85a1feff297322db51
7 git commit init/main.c -m "print that the kernel is starting up"
```

```

8 git format-patch master --cover-letter -o ../patches-staging
9 git diff master
10 ./scripts/checkpatch.pl ../patches-staging/0001-print-that-the-kernel-is-starting-up.patch
11 total: 0 errors, 0 warnings, 8 lines checked
12 ../patches-staging/0001-print-that-the-kernel-is-starting-up.patch has no obvious style
    problems and is ready for submission.

```

Is it ready for shipping?

Many people would frown now asking for some serious testing before submitting, but since GNU/Linux supports more architectures and more hardware/drivers than any other operating system you don't have the equipment and the time to test it all. GNU/Linux runs on 470 of the Top 500 supercomputers in 2010 [20] and is one of the most popular operating systems for mobile phones (Moblin [21], Android [22], MeeGo [23], OpenMoko [24],...) There is a GNU/Linux Test Project [25] which is used by a few people, but there is no official GNU/Linux kernel test suite. You have to rely on the community for exhaustive tests. Even scripts/checkpatch.pl says that *"the patch has no obvious style problems and is ready for submission"*.

Audited code

Let's come back a bit to the code review/audited code facilities provided by git. First we'll see how our own changes are documented by git:

```

1 cd $ELDK_PREFIX/eldk-4.2-ppc_4xx/usr/src/linux-2.6-denx-latest
2 git log

```

That's what git returns:

```

1 commit 23c0a904887258406922fbd9a1f216f74900c71d
2 Author: Robert Berger <git.rber@gmail.com>
3 Date: Mon Apr 19 21:18:26 2010 +0300
4     fixed checkpatch WARNING
5 commit 681e18b8925d026d266919d44736d88859efef4c
6 Author: Robert Berger <git.rber@gmail.com>
7 Date: Mon Apr 19 20:06:34 2010 +0300
8     my first kernel patch

```

In order to demonstrate how git tracks who did what let's try to find if and what Thomas (Gleixner) did with the file which was patched:

```

1 git blame init/main.c
2 git blame init/main.c | grep Thomas

```

git tells us that Tom added an #include:

```

1 906568c9 (Thomas Gleixner 2007-02-16 01:28:01 -0800 47) #include <linux/tick.h>

```

Who did write/commit/review/sign-off this commit?

```

1 git show --pretty=full 906568c9

```

Looks like a couple of people saw the commit:

```

1 commit 906568c9c668ff994f4078932ec6ae1e3950d1af
2 Author: Thomas Gleixner <tglx@linutronix.de>
3 Commit: Linus Torvalds <torvalds@woody.linux-foundation.org>
4     [PATCH] tick-management: core functionality
5     Signed-off-by: Thomas Gleixner <tglx@linutronix.de>
6     Signed-off-by: Ingo Molnar <mingo@elte.hu>
7     Cc: john stultz <johnstul@us.ibm.com>

```

```
8 Cc: Roman Zippel <zippel@linux-m68k.org>
9 Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
10 Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>
```

Every line in the GNU/Linux kernel is at least reviewed by 2 people and as we just saw git enables us to trace who modified last/reviewed each line.

Ship it?

The patch git created already looks like an email in your mailbox.

```
1 From 05961bc606cedc075e4f8c9ef7ef248956a91337 Mon Sep 17 00:00:00 2001
2 From: Robert Berger <git.rber@gmail.com>
3 Date: Sun, 19 Sep 2010 23:49:33 +0300
4 Subject: [PATCH] print that the kernel is starting up
5
6
7 Signed-off-by: Robert Berger <git.rber@gmail.com>
8 ---
9  init/main.c | 1 +
10 1 files changed, 1 insertions(+), 0 deletions(-)
11
12 diff --git a/init/main.c b/init/main.c
13 index 94ab488..5b36538 100644
14 --- a/init/main.c
15 +++ b/init/main.c
16 @@ -538,6 +538,7 @@ asmlinkage void __init start_kernel(void)
17     char * command_line;
18     extern const struct kernel_param __start__param[], __stop__param[];
19
20 + printk(KERN_INFO "The kernel is starting up!\n");
21     smp_setup_processor_id();
22
23     /*
24     --
25 1.7.0.4
```

Mail clients/servers

Be careful which email client you use and how. For further information you might want to have a look at Documentation/emailclients. GUI based email clients may destroy the patch. You might want to try with gmail and mutt. Note that even if you do the right thing with the email client mail servers may destroy the patch. With the right email setup git can send out the patch.

```
1 [sendemail]
2     chainreplyto = false
3     smtpserver = smtp.gmail.com
4     smtpserverport = 587
5     smtpencryption = tls
6     smtpuser = git.rber@gmail.com
7     confirm = auto
```

It's time to send out the patch to ourselves first, receive it, apply it and check if it works.

Send out the patch

```
1 cd $ELDK_PREFIX/eldk-4.2-ppc_4xx/usr/src/linux-2.6-denx-latest
2 git branch
```

```
3 git checkout master
4 git checkout -b kernel-patch-from-mutt
5 git send-email --to git.rber@gmail.com ../patches-staging/*.patch
```

Receive and save patch

```
1 mutt
2 s ../patches-from-mutt/0001-my-first-kernel-patch.patch
```

Please note that we receive the cover letter which we created before. This should be done to announce that a series of patches follows.

Apply the patch

```
1 git am ../patches-from-mutt/0001-my-first-kernel-patch.patch
2 git diff master
```

Test it

Rebuild the kernel as described above and check if everything still builds and works as expected.

Ship it!

Now you are ready to email the patch out to the community. Go ahead and contribute, but don't forget to CC the appropriate people.

Conclusion

Hopefully you got some idea about the process involved in creating and submitting a GNU/Linux kernel patch. Now you need to try it for yourself. As already mentioned the staging tree is a great place to get your feet wet. Work with the community from the early stages (design) on and submit your code early as well - don't wait until it's perfect.

Why would your boss be interested in having you contribute high quality code to the community?

- the maintenance costs will be smaller
- the design quality will be higher
- the code quality will be higher

You might not want to tell your boss, but why should you contribute high quality code to the community?

- the cosy feeling of being a respected "member"
- plenty of new job opportunities
- better pay

References

- [1] "Andrew S. Tanenbaum" <http://www.few.vu.nl/~ast/>
- [2] "Linux is obsolete" <http://www.educ.umu.se/~bjorn/mhonarc-files/obsolete/index.html>
- [3] "Biography of Linus Torvalds" <http://events.linuxfoundation.org/lc09p2>
- [4] "Biography of Sergey Mikhailovich Brin" http://en.wikipedia.org/wiki/Sergey_Brin
- [5] "Google operates what is probably the world's largest Linux cluster that puts many supercomputing centers to shame." <http://www.n3labs.com/pdf/google-class11.pdf>
- [6] "Kernel history"
http://www.kernel.org/pub/linux/kernel/people/gregkh/kernel_history/
- [7] Barry W. Boehm et. al., *Software Cost Estimation with Cocomo II*, ISBN: 978-0130266927, Prentice Hall (August 11, 2000).
- [8] Barry W. Boehm, *Software Engineering Economics*, ISBN: 978-0138221225, Prentice Hall (November 1, 1981).
- [9] "Linux 2.6.30" http://kernelnewbies.org/Linux_2_6_30
- [10] "Kernel cost" <http://www.dwheeler.com/essays/linux-kernel-cost.html>
- [11] "Linux kernel R&D worth over 1bn euros" http://www.theregister.co.uk/2010/02/24/linux_kernel_randd_estimate_u_of_oviendo/
- [12] "mainline, stable, linux-next" <http://www.kernel.org/>
- [13] "subsystem" <http://git.kernel.org/>
- [14] "Bitkeeper reverse engineered" <http://lwn.net/Articles/132938/>
- [15] "After controversy, Torvalds begins work on git" <http://www.infoworld.com/t/platforms/after-controversy-torvalds-begins-work-git-721>
- [16] "Why the 'git' name?" https://git.wiki.kernel.org/index.php/GitFaq#Why_the_.27git.27_name.3F
- [17] "Kilauea Board" <http://www.appliedmicro.com/Embedded/Downloads/download.html?cat=1&family=18>
- [18] "Embedded Linux Development Kit" <http://www.denx.de/wiki/DULG/ELDK>
- [19] "Das U-Boot – the Universal Boot Loader" <http://www.denx.de/wiki/U-Boot>
- [20] "Nearly every supercomputer runs Linux" <http://www.itwire.com/business-it-news/technology/39471-nearly-every-supercomputer-runs-linux>
- [21] "Moblin" <http://moblin.org/>
- [22] "Android" <http://www.android.com/>
- [23] "MeeGo" <http://meego.com/>
- [24] "Openmoko" <http://www.openmoko.com/>
- [25] "Linux Test Project" <http://ltp.sourceforge.net/>

Author



Robert Berger consults and trains people all over the globe on a mission to help them create better embedded software. His specialties are trainings and consulting in the broad field of Embedded Software from small Real-time Systems to multi-core Embedded Linux. You can contact Robert here: robert.berger@ReliableEmbeddedSystems.com