Linux Kernel Maintainers

Greg Kroah-Hartman
gregkh@linuxfoudation.org

Why are they so grumpy?

What you can do to avoid this.

What maintainers owe you.

2,833 developers
373 companies

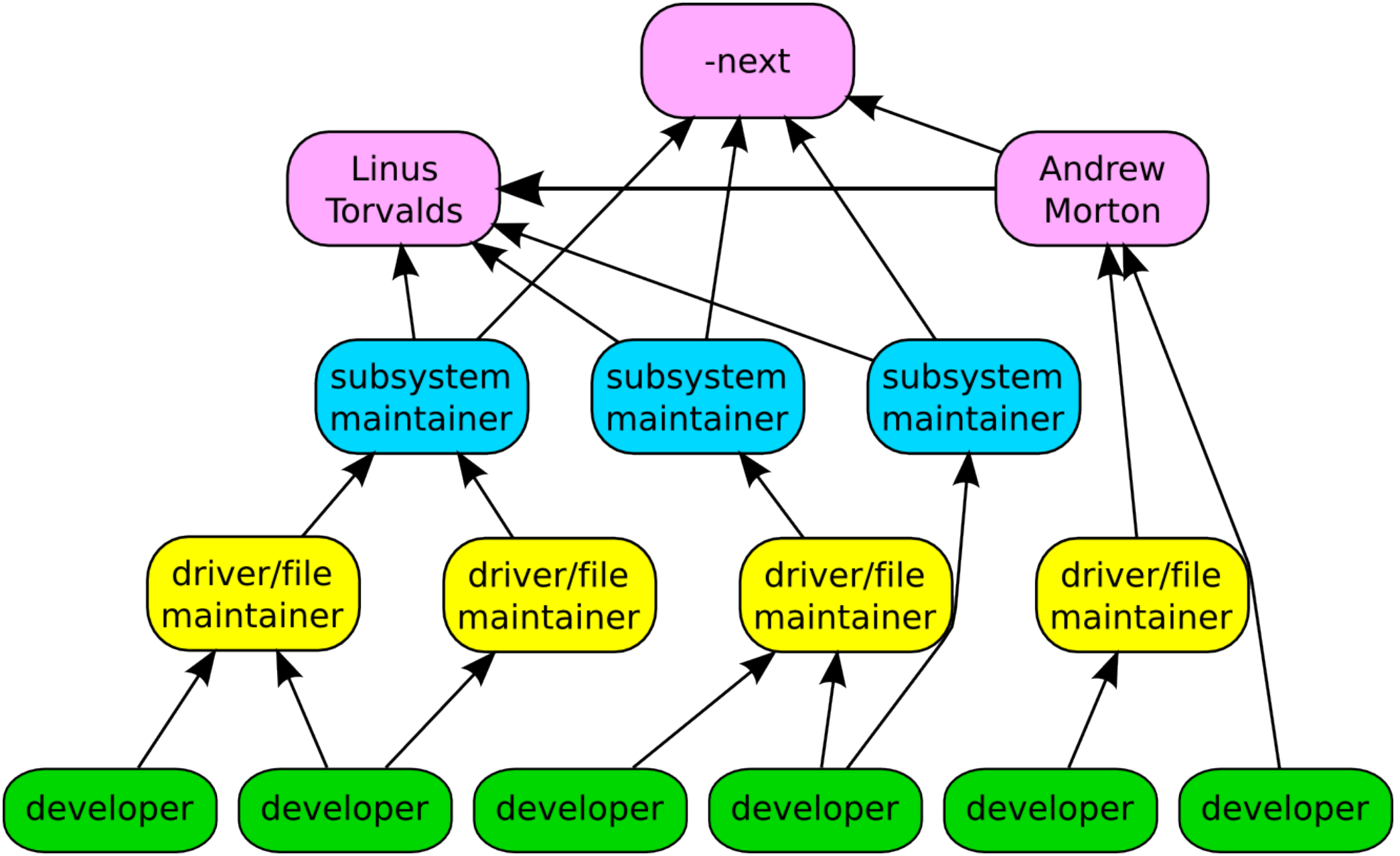
Kernel releases 3.0.0 – 3.4.0
May 2011 – May 2012

5.79 changes per hour

Kernel releases 3.0.0 – 3.4.0
May 2011 – May 2012

7.21 changes per hour

3.4.0 release


-next
Linus Torvalds
Andrew Morton
subsystem maintainer
subsystem maintainer
subsystem maintainer
driver/file maintainer
driver/file maintainer
driver/file maintainer
driver/file maintainer
developer
developer
developer
developer
developer
developer

Patches I received in the past 2 weeks

Patches I received in the past 2 weeks

487

Subject: [PATCH 48/48] ...

15 patch series, no order given

Patches 1, 3-10

“Signed-off-by:” in signature

Signature saying email was confidential

Tabs were converted to spaces

Leading spaces removed

diff in non-unified format

Patch created in driver directory

Patch created in /usr/src/linux-2.6.32

Made against different tree

Wrong coding style

Wrong coding style, and acknowledged it

Would not compile

Broke the build on patch 3/6

Broke the build on patch 3/6 and fixed it on 6/6

Broke the build on patch 5/8

Broke the build on patch 5/8

Contained note that fix would be sent later

Patches that had nothing to do with me

1 patch, 450kb big (4500 lines added)

Obviously wrong kerneldoc

This was a calm two weeks

It is in my self-interest to ignore your patch

Give me no excuse to reject your patch

Proper coding style

scripts/checkpatch.pl clean

Sent to proper people and lists

Sent to proper people and lists

scripts/get_maintainer.pl

Proper Subject:

Proper changelog comment

Description of WHY it is needed

Small incremental change

“obviously” correct

Which tree it was made against

If multiple patches, state the order

Has to build properly

Make sure it works, if possible

Don't ignore review comments

Don't resend without saying why

What I will do for you:

Review your patch within 1-2 weeks

Offer semi-constructive criticism

Let you know the status of your patch

“Publicly making fun of people is half the fun of open source programming.

In fact the main reason to eschew programming in closed environments is that you can't embarrass people in public.”

– Linus Torvalds

“Publicly making fun of people is half the fun of open source programming.

In fact the main reason to eschew programming in closed environments is that you can't embarrass people in public.”

– Linus Torvalds

github.com/gregkh/presentation-maintainer

Linux Kernel Maintainers

Greg Kroah-Hartman
gregkh@linuxfoudation.org

When I first started writing this talk, it quickly turned into one big long rant. I ended up listing all of the different problems that I had with patches that people had sent me over the past few years.

While this would have been a very fun and cathartic talk for me, I figured that you all just watching me complain for 30 minutes wouldn't be the most intertaining thing, so I figured I would try to tone it down.

Why are they so grumpy?

What you can do to avoid this.

What maintainers owe you.

So, let's talk about the main problem that people seem to have with Linux kernel maintainers, why are they so grumpy? Hopefully by the end of this talk, you will have an idea of why this always happens, and what you can do to avoid having that anger be directed at you.

Also, I'm going to cover what you should expect from a good kernel maintainer, so if you are a maintainer, here's something that developers can use to get back at you, and me, as I figure it's only fair.

I am going to complain a lot in this talk. Please don't get the impression that I don't like doing this type of work. I love it. It's the best job in the world that I've ever had, and I can't think of anything that I would rather be doing.

2,833 developers
373 companies

Kernel releases 3.0.0 – 3.4.0
May 2011 – May 2012

This makes the Linux kernel the largest contributed body of software out there that has been created..

This is just the number of companies that we know about, there are more that we do not, and as the responses to our inquiries come in, this number will go up.

5.79 changes per hour

Kernel releases 3.0.0 – 3.4.0
May 2011 – May 2012

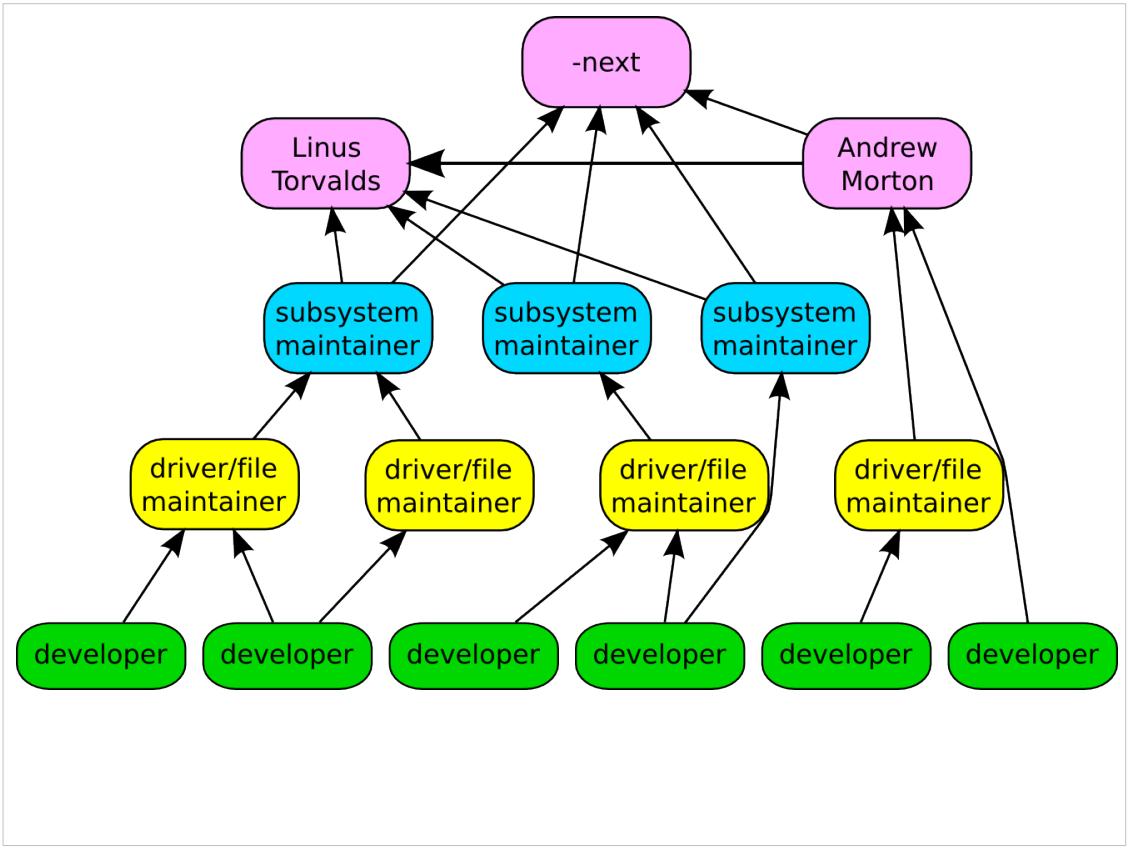
For that year of development, we went at this rate, 24 hours a day, 7 days a week. This is up from last year, which was at 5.2 or so, so we are increasing, which is scary, right?

7.21 changes per hour

3.4.0 release

This past 3.4 release was the fastest we have ever created. That number shows just how well the Linux kernel development model is working. We are growing in developers and in how fast we are developing overall.

Now this is just the patches we accepted, not all of the patches that have been submitted, lots of patches are rejected, as anyone who has ever tried to submit a patch can attest to.



Here's a picture of our development model, in a simplified form.

We have about 3000 different developers. They make a patch, and send it through email to the file/driver maintainer. We have about 700 different maintainers listed in the kernel tree at the moment. That maintainer reviews it, and if they accept it, they forward it on to the subsystem maintainer. We have around 130 different subsystem maintainers at the moment.

Those maintainers have public kernel trees that all get merged into the linux-next release every day. Then, when the merge window opens up, the subsystem maintainers send their stuff to Linus.

Patches I received in the past 2 weeks

So, let's look at one of these subsystem maintainers. I maintain the USB, driver core, tty, staging, and a few other various parts of the Linux kernel.

These past 2 weeks is the timeframe when we had our big merge window, when all of the subsystem maintainers sent patches off to Linus. During this time frame, no core kernel developer sends new stuff to subsystem maintainers, as they know they are busy, and nothing that gets sent can really be looked at until after the merge window closes.

So, almost all of the patches I got in the past 2 weeks were not from developers that do a whole lot of kernel work, nor were the, for the most part, large patches with new things being proposed for the kernel.

Patches I received in the past 2 weeks

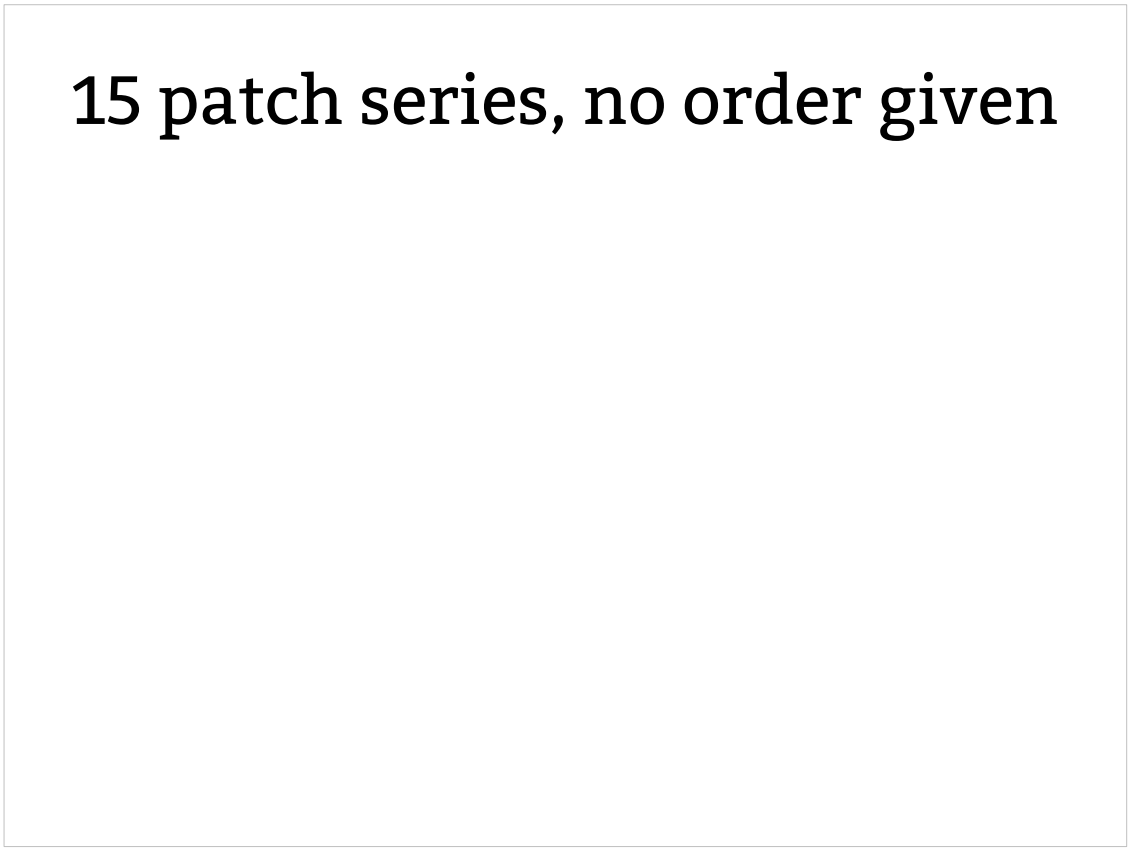
487

Yeah, that's the number of patches I got during the "slow" period of the kernel development cycle. This does not include the number of messages around those patches as other developers commented on them, or other various things about those patches (like "have you applied my patch yet?" messages.)

Now the large majority of these patches at first glance look just fine. But I took a closer look at them, and here's a short list of the problems in the patches that were sent to me.

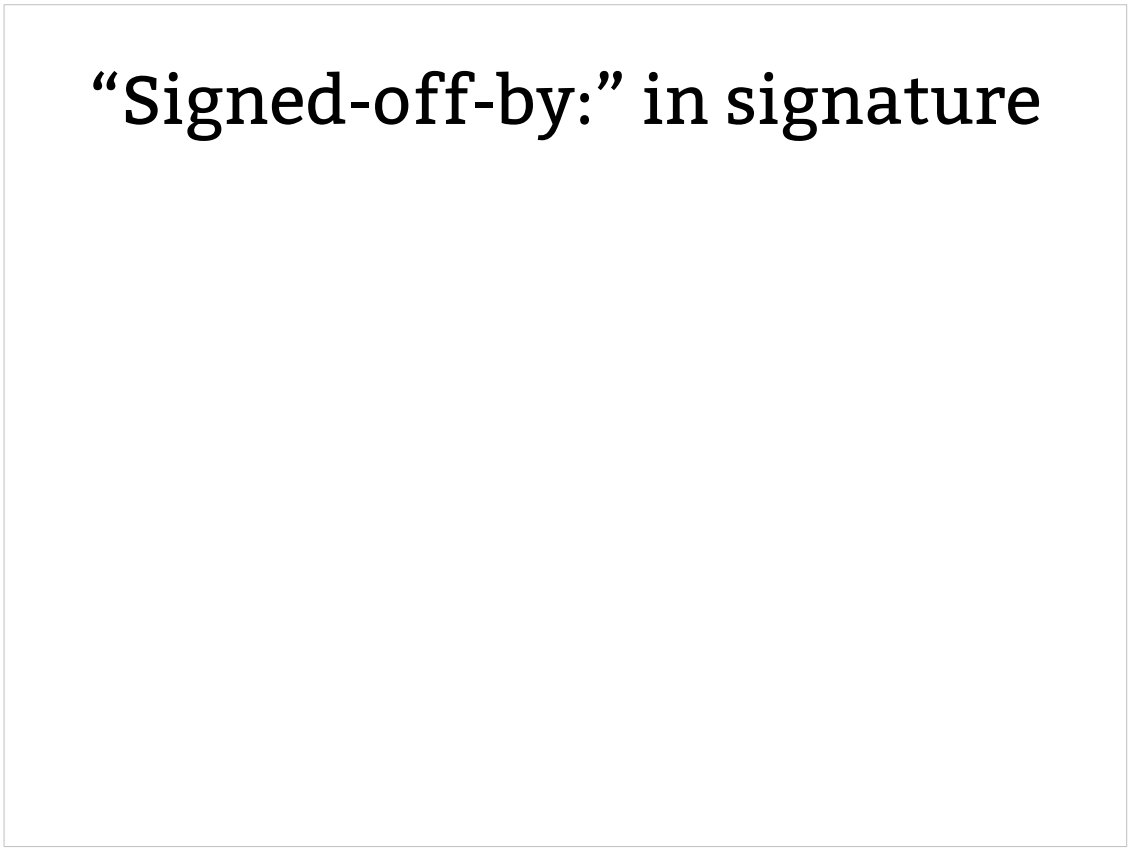
Subject: [PATCH 48/48] ...

There were no 47 previous patches sent.

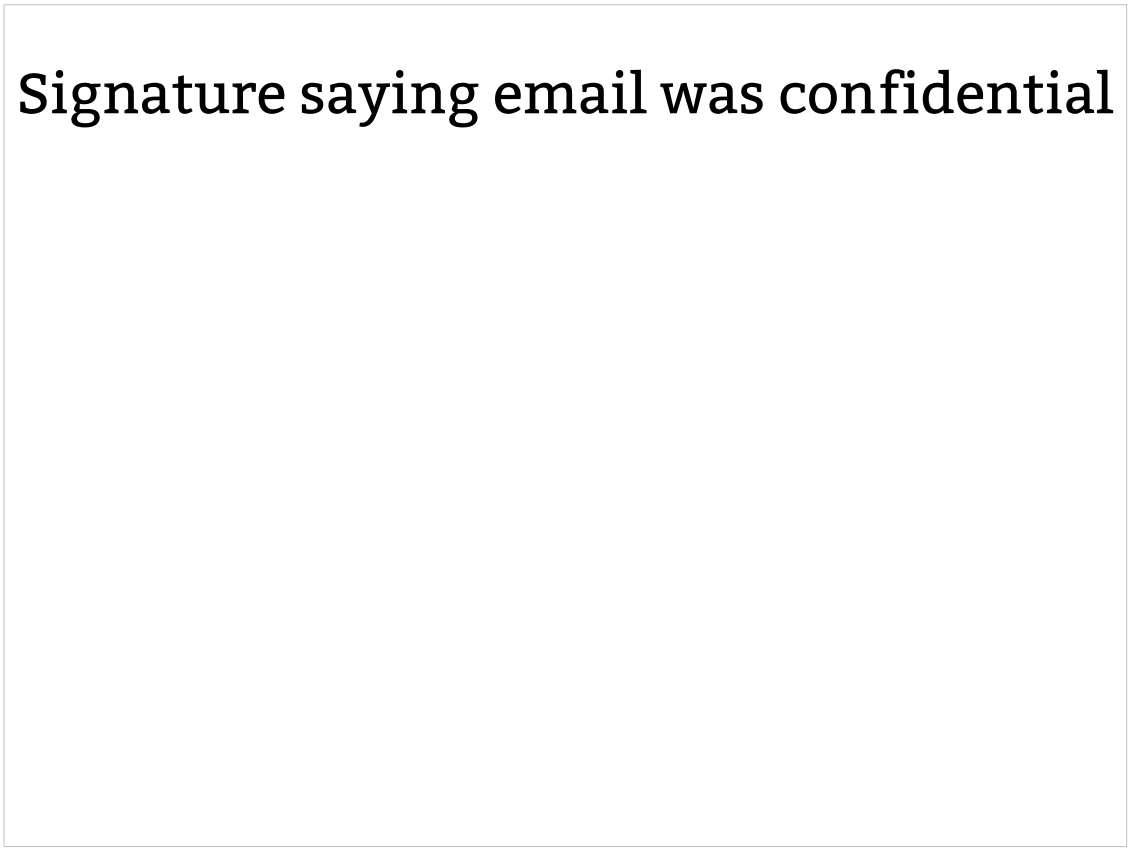


Am I supposed to guess?

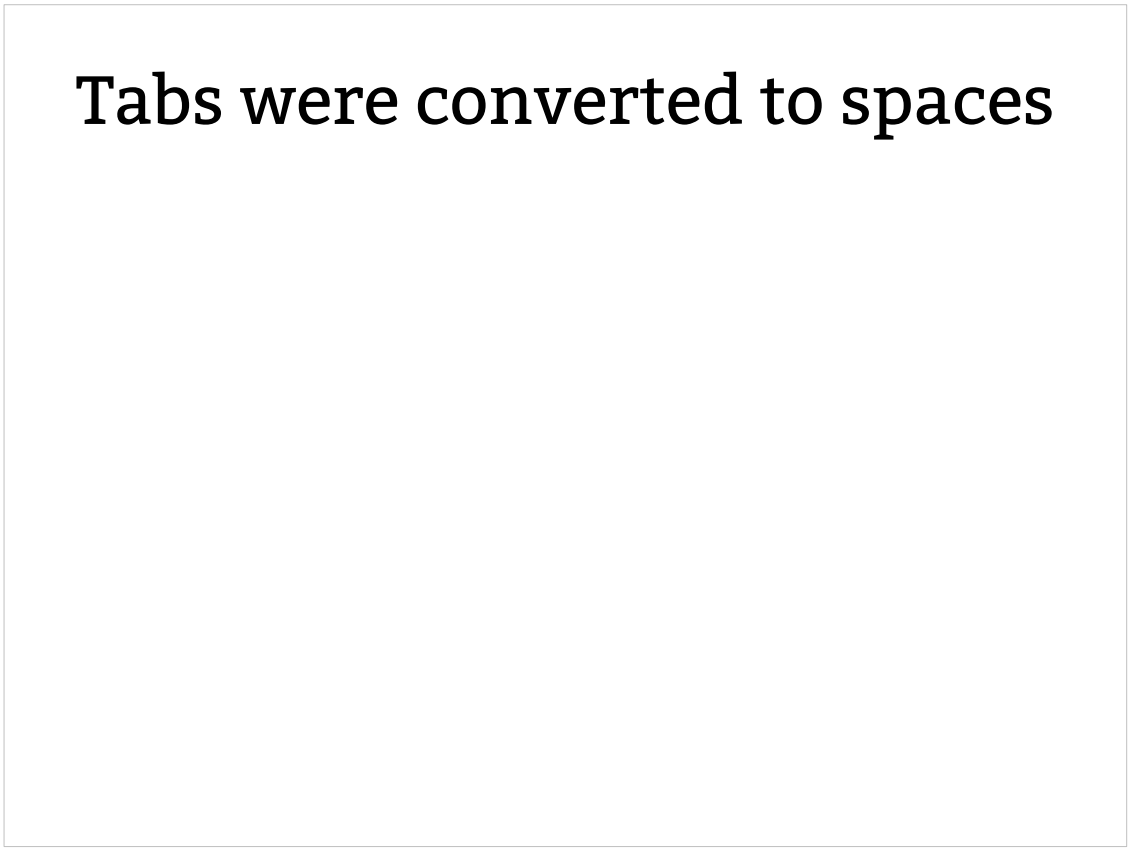
Patches 1, 3-10



This would require me to hand edit the patch before I could apply it.



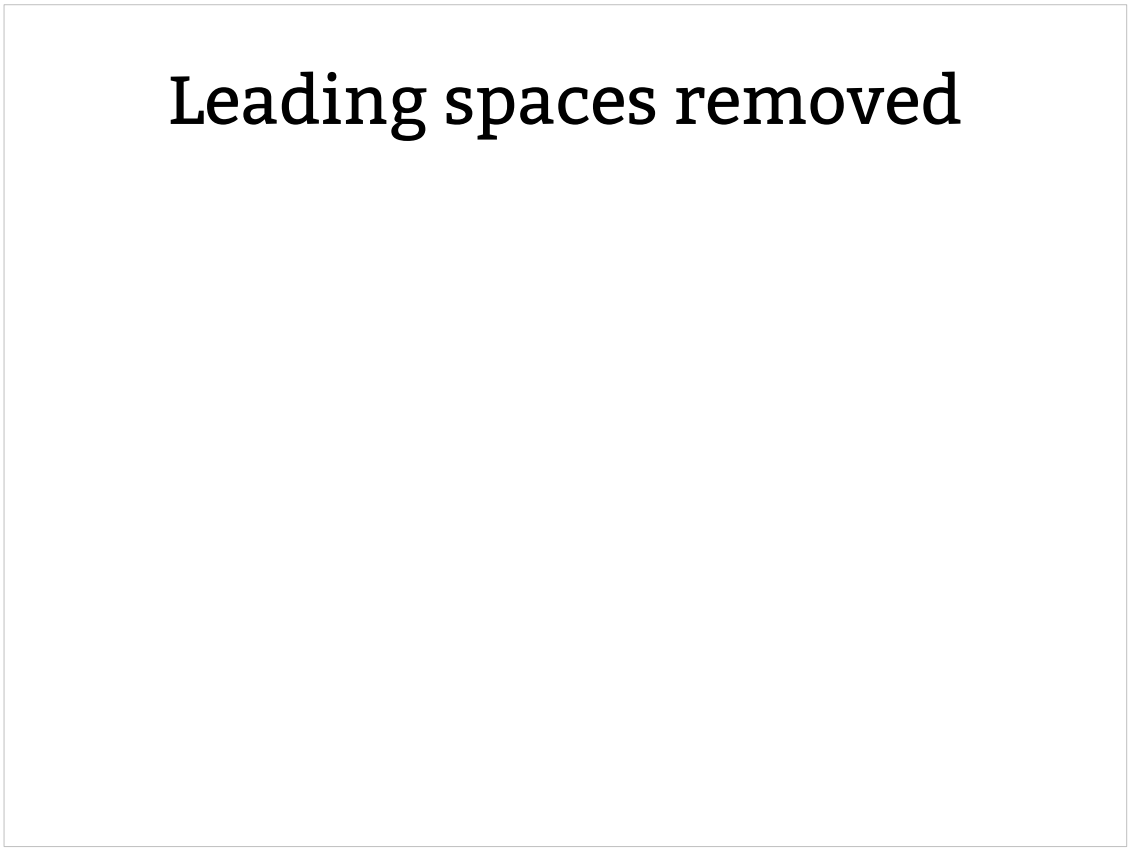
That kind of goes against how you are supposed to be sending Linux kernel patches out to the world.



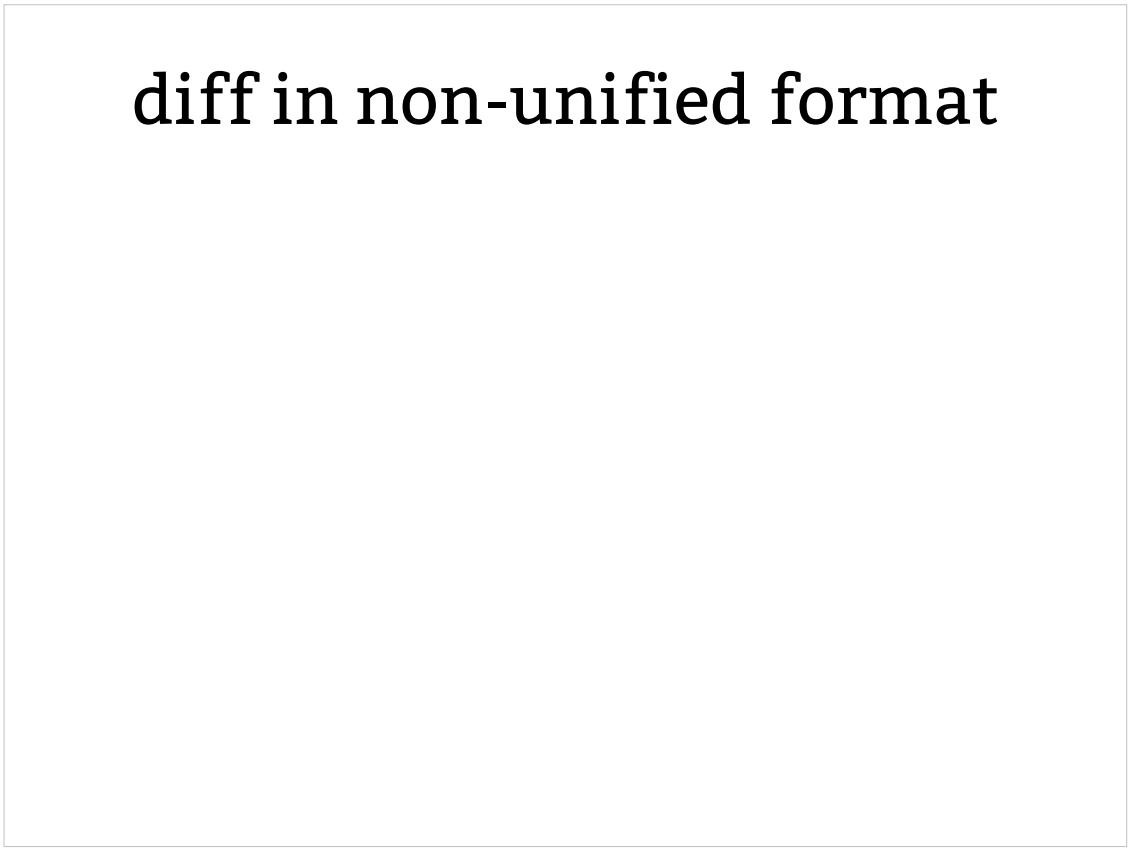
This makes applying the patch impossible.

Exchange does this for you, if you are working for a corporation that has an Exchange server, do what IBM, Intel, and Microsoft have done in order to be able to contribute to Linux kernel development, have a Linux box somewhere in the corner that your developers use as a mail server to send patches out from.

Huawei is the only company that I know of that successfully sends kernel patches through an Exchange server, which is amazing, I really don't know how they do it.



This also makes applying the patch impossible. I end up editing a lot of patches by hand, cursing all the while, just to get them to apply because of broken email servers and clients.

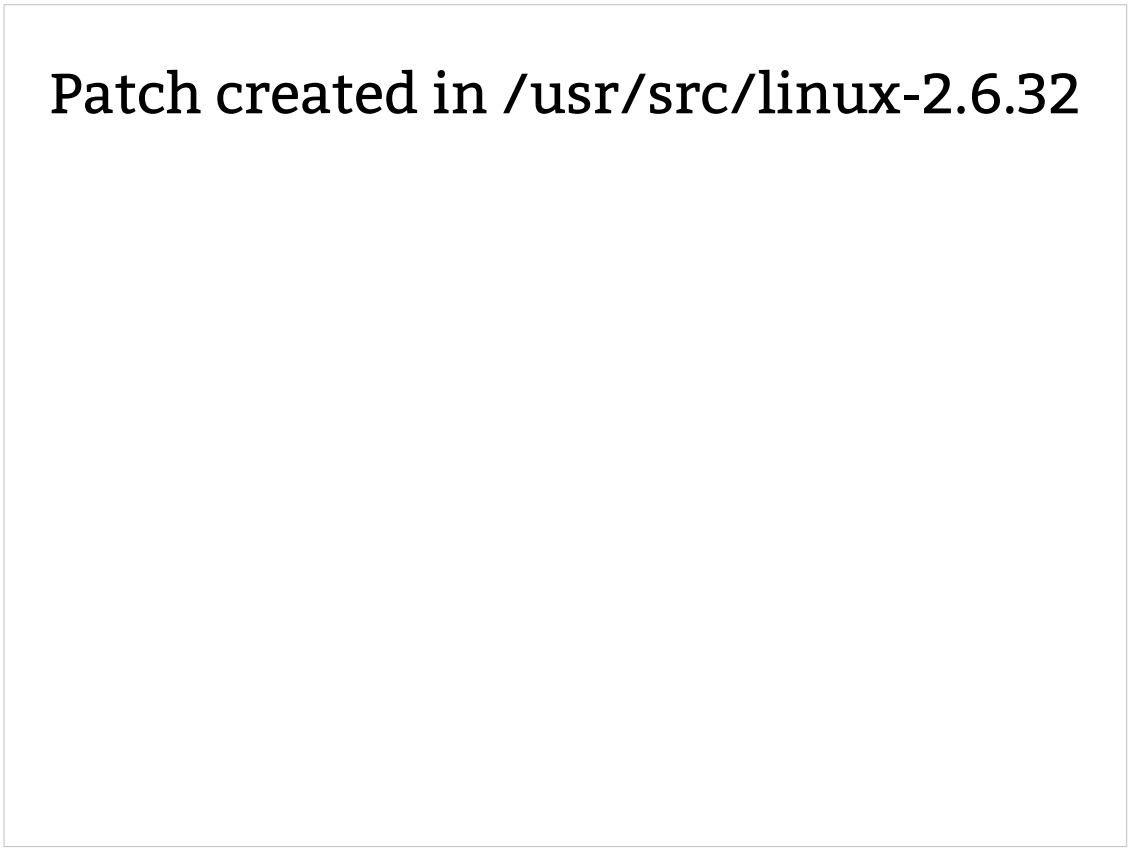


I honestly didn't know that diff could still create output in this format anymore, I assumed that as no one ever found it useful, it wasn't used anymore.

Patch created in driver directory

Patches need to be created in the root of the kernel source tree, as that's where I have to be in order to apply them properly.

This seems to happen a lot to first-time patch submitters, it's a very common problem.

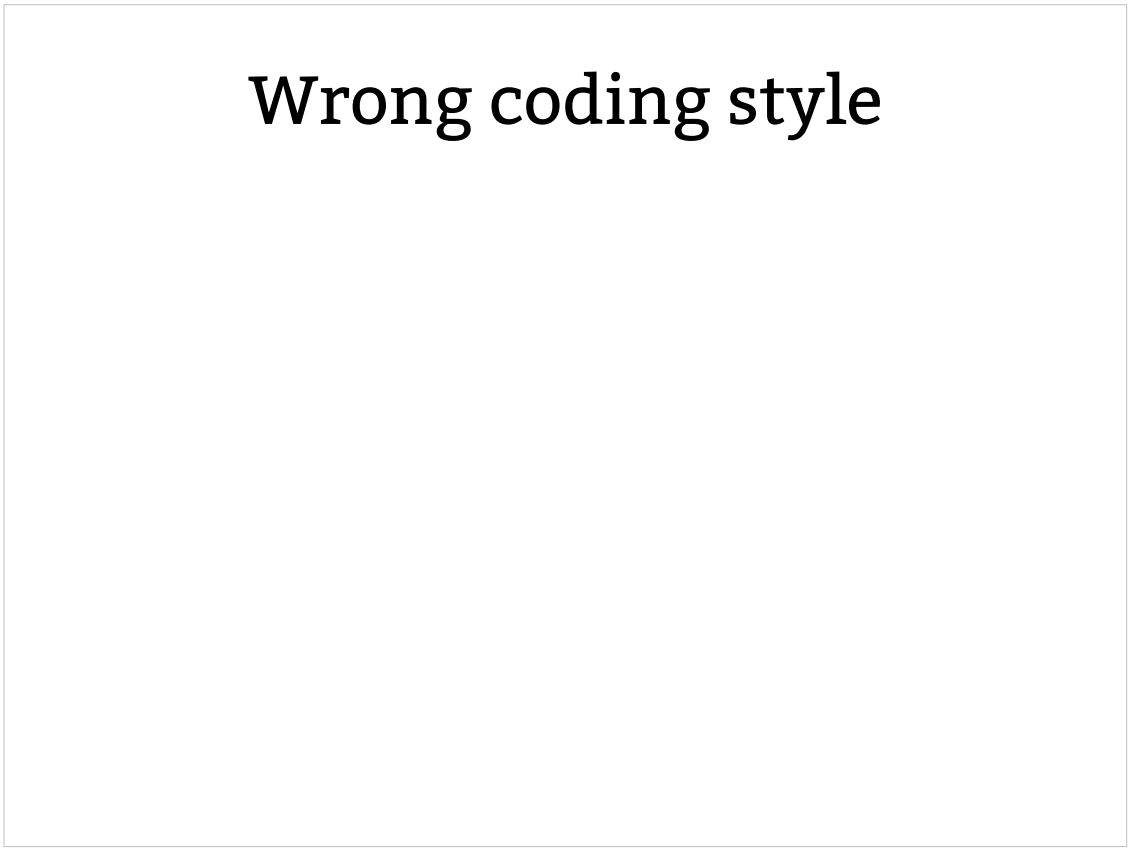


How many different problems can you see here in just this one example?

Old and obsolete kernel version, full path to root, developer doing kernel work as root, probably more.

Made against different tree

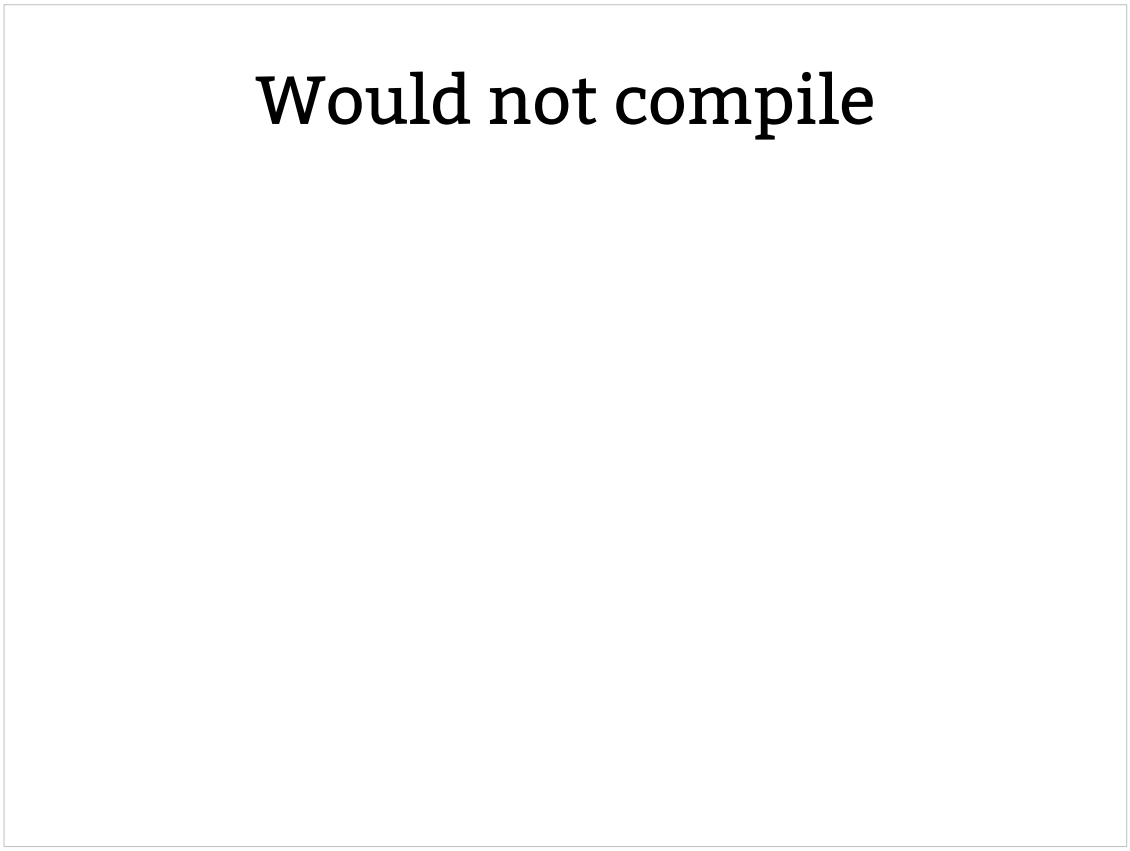
Someone made a patch against the scsi subsystem development tree when sending me a USB patch. Why they thought that was a good idea I have no idea.



There's no excuse for doing something like this anymore, we have automated tools that fix this up for you.

Wrong coding style, and acknowledged it

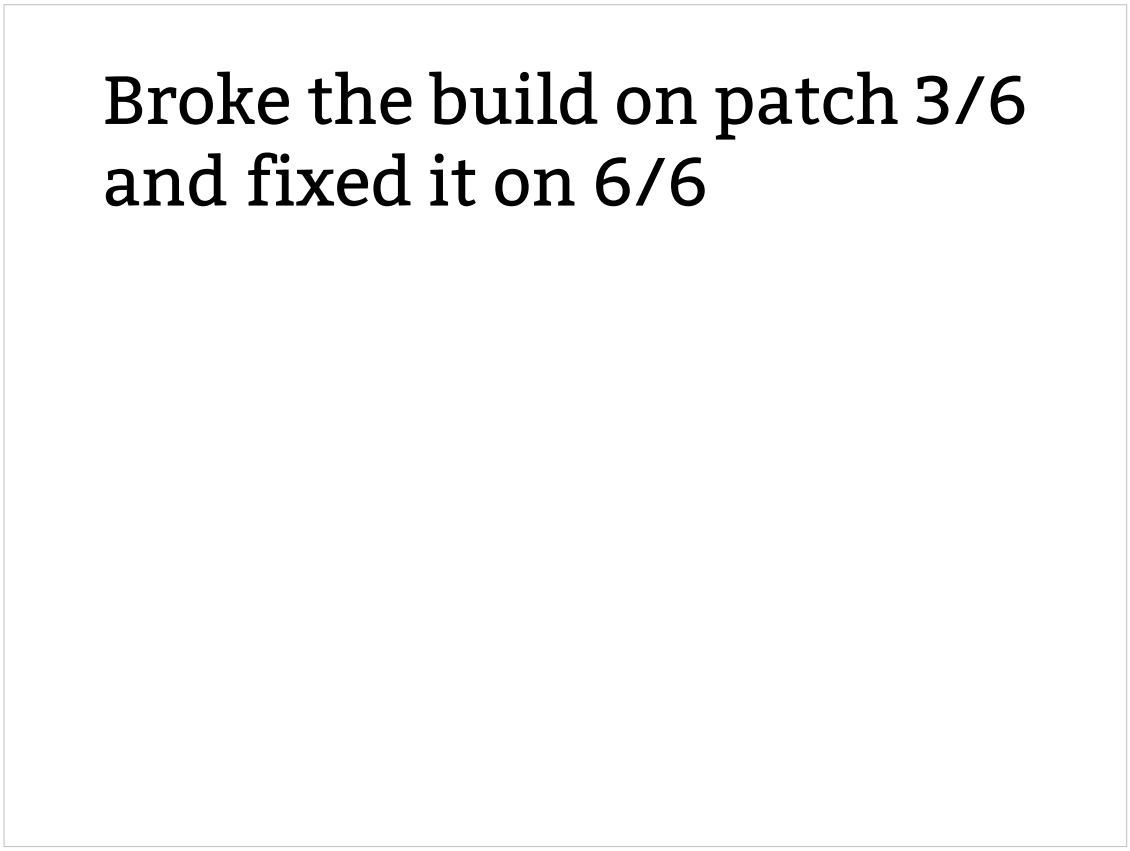
At least in this patch, the author knew they were doing something wrong, It's just that they thought they were more important than the 3000 other kernel developers and didn't have to play by the rules of everyone else.



Just looking at the patch it was obvious that it had never been compiled, and sure enough, the compiler spit out a bunch of errors.

Broke the build on patch 3/6

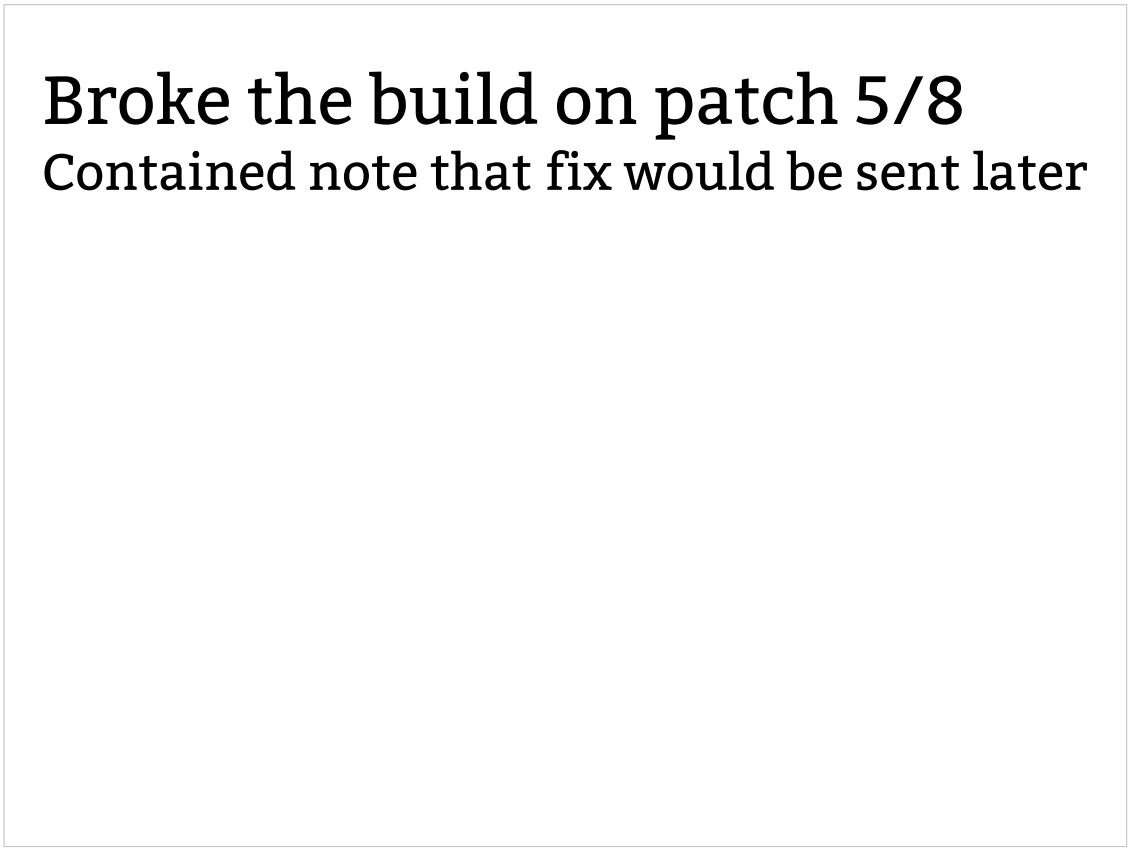
This was a series of patches, and the build broke on the 3rd patch that was applied.



But, I looked closer, and the developer at least realized this, and fixed it in their last patch in the series, thinking that all was now good, as it didn't really matter that for the past 3 patches, the build was broken.

Broke the build on patch 5/8

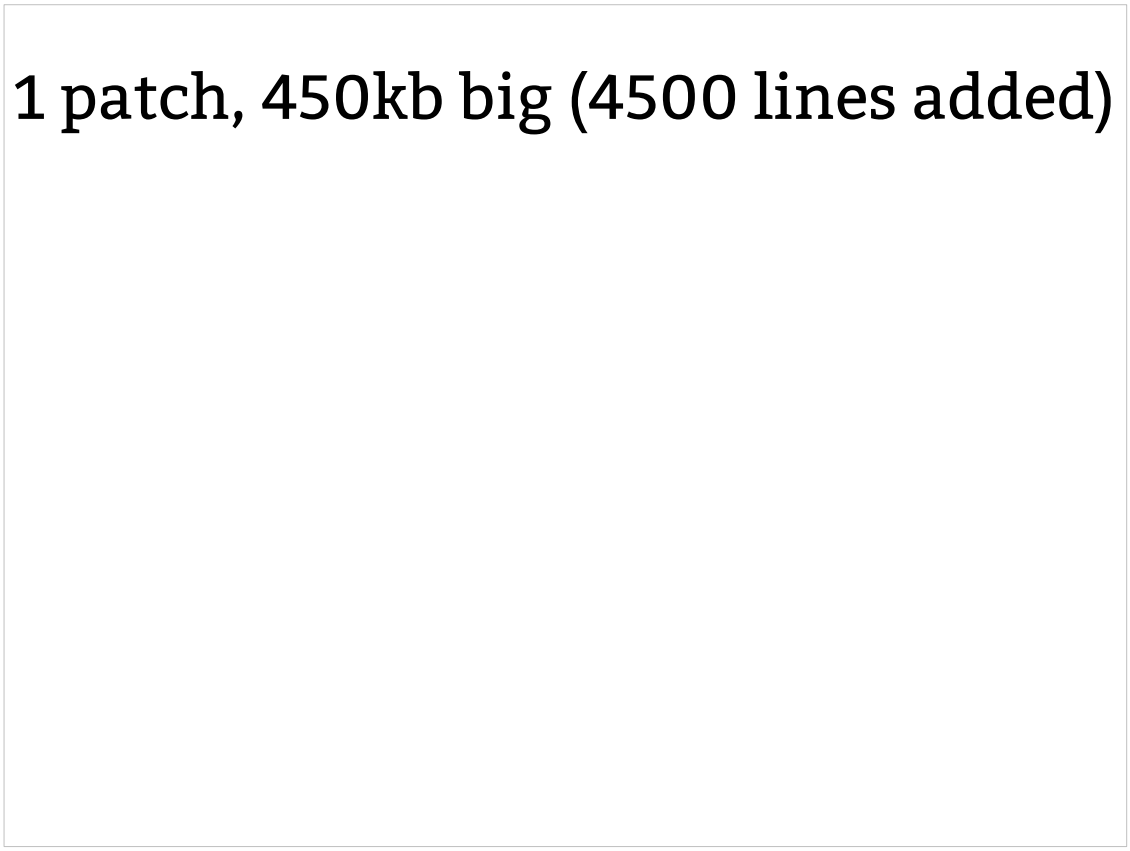
There was another patch series that also broke the build in the middle of it.



But this one was better, it had a note saying that they knew the build was broken, and they would fix it up later, at some unknown time in the future, but these 8 patches should be accepted now anyway.

Patches that had nothing to do with me

Now I know I maintain a lot of different parts of the kernel, but for some reason someone sent me a patch for the x86 core code, copied to no one else, thinking that I was the one that could accept it.



Luckily, another developer told the author that this was too big and needed to be broken up into smaller pieces before anyone would review it. And then, three different developers went and reviewed it anyway, so I don't think the author learned that lesson at all.

Obviously wrong kerneldoc

kerneldoc is the format that you can write comments in the code and get them included in the kernel api documentation that is automatically generated. When you get the format of it wrong, the tools will tell you.

Here was someone who was trying to write documentation, but got the format wrong, and hadn't even run the tools to see if it was generated properly.

This was a calm two weeks

Now, I'm not asking you to take pity on me, just realize that this is the level of incompetence that every single one of those 700 developers encounter all the time.

So when you think we are acting grumpy, remember, how would you act if you had to deal with this all of the time?

Let's get back to what the goal is here. You want to create a patch that is accepted as it does something that you want to do in Linux. The maintainer wants to reject it.

It is in my self-interest to ignore your patch

Seriously. It's easier for the maintainer to not accept your code at all. To accept it, it takes time to review it, apply it, send it on up the development chain, handle any problems that might happen with the patch, accept responsibility for the patch, possibly fix any problems that happen later on when you disappear, and maintain it for the next 20 years.

That's a lot of work that you are asking someone else to do on your behalf. You are asking someone who doesn't usually work for your company, who probably lives in a different country, who you have never met in person, to assume responsibility for your work, and to do extra work on top of the normal work they do in the kernel every day.

So you can see how it's in my interest to ignore your patch. And it's in your interest to keep me from ignoring it, because you want it accepted.

Give me no excuse to reject your patch

So your goal is, when sending a patch, is to give me NO excuse to not accept it. To make it such that if I ignore it, or reject it, I am the one that is the problem here, not you.

What can you do to keep me from rejecting your patch outright

.

First off, don't do any of the things I listed above, that's obvious, right? But that's a "do not do" list, how about a list of what to do:

Proper coding style

This is documented, there should not be any reason to ever get this wrong.

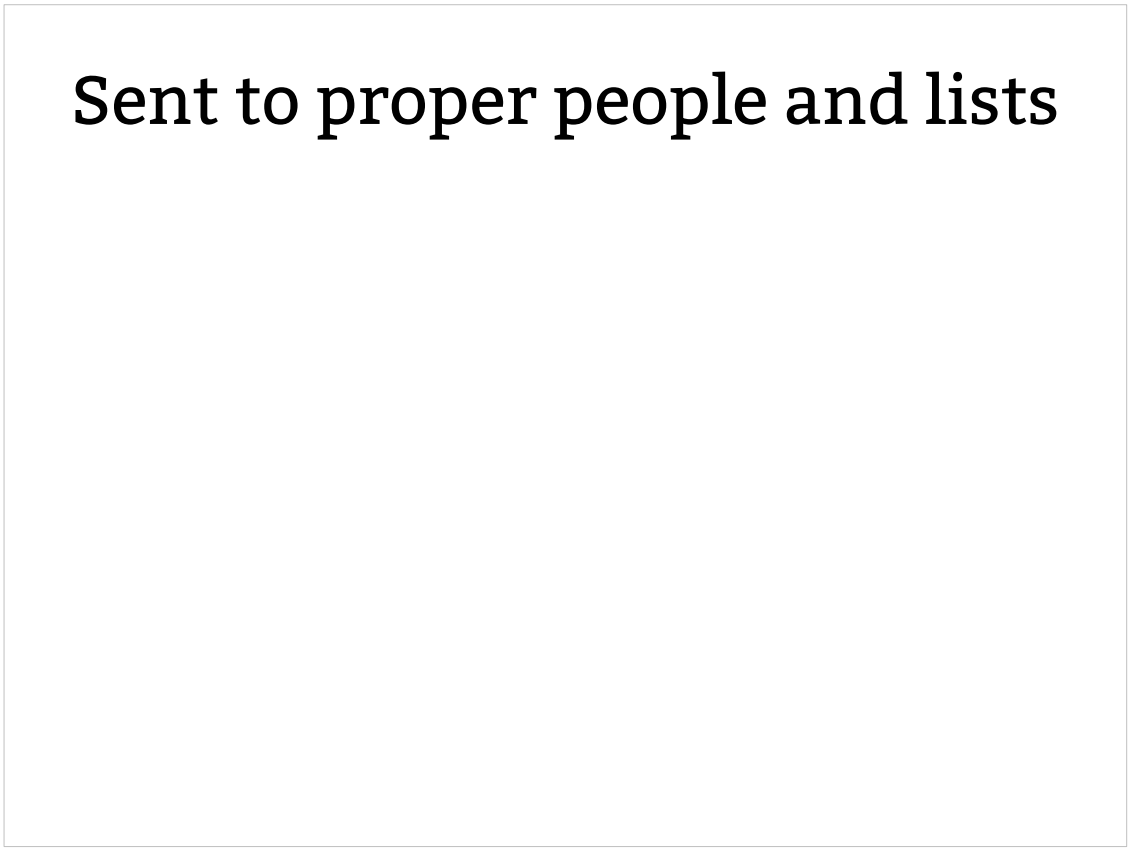
Or to think that you are above following the rules, that's just asking for the patch to be rejected.

scripts/checkpatch.pl clean

We even have a tool that automatically checks your patch to ensure that you got the coding style correct, and that other common problems are avoided.

If you don't run this tool, the maintainer will, and will get mad when it finds problems that you should have solved before sending it to them.

Sent to proper people and lists



I have an email bot that if you ever send a patch to only me, and not any mailing list, instantly rejects it and tells you to resend it and copy the proper people and mailing lists.

Linux kernel development is done in public, not private, and it doesn't scale to send patches or emails to only individual developers. We all have subsystem-specific mailing lists, use them, that way other people can review your patches, and comment on them, and you don't overburden the individual subsystem maintainers any more than they already are.

Sent to proper people and lists

scripts/get_maintainer.pl

Look, we even have a tool that you run on your patch, and it digs through the MAINTAINERS file and the git history, and figures out who to send the patch to, and what mailing lists to copy at the same time.

Use it.

Proper Subject:

Make the subject of your patch something that makes sense.

Don't send me a 20 patch series that for every individual patch says, "Fixes problems in driver" like some people have done in the past.

Proper changelog comment

In the body of the email, describe the patch, what it does. And most importantly:

Description of WHY it is needed

Too many times we see patches that say exactly what the patch does. Which is stupid because we know how to read code, what we want to know is why the change is being made, and from that we can determine if it really is needed or not.

Small incremental change

Patches are not supposed to be big huge rewrites of things. That's not how we do development. You need to make each patch a small one-item change.

Break your larger change up into a set of small, individual, steps. Like your math professor said, "show your work". We want to see all the steps you make along the way to complete your larger goal.

“obviously” correct

Make it the patch is so simple and obvious that I would be foolish to reject it. I need to read it and say, "of course, I can't belive we didn't do that in the past, how stupid we never did this before!"

Which tree it was made against

If you create a patch against a different development tree than the person you are sending it to, let them know. If you made it against an obsolete vendor enterprise kernel release, tell them. Don't make them guess.

If you make me guess, I will guess wrong, that's just the way it goes.

If multiple patches, state the order

Number your patches, don't rely on my email client receiving them in the same order that you sent them in. I guarantee that will not happen properly.

git send-email does this correctly for you, use that to send your patches out.

Has to build properly

At least build the change before you send it to me. Because if it breaks the build, it just makes me more likely to not want to apply anything else you send me in the future, as you are just wasting my time.

Make sure it works, if possible

If you have the hardware, test the patch. Now that isn't always possible, and that's fine, we make changes to drivers for hardware that we don't have access to all the time, which seems to surprise a lot of people.

Go back to that "obviously correct" item, if you don't have the hardware, stick to that rule and you will be fine.

Don't ignore review comments

Lots of time I see patches sent out on Friday afternoon, and then the author disappears on a 2 week vacation. So, when I spend the time reviewing the patches, I get back a vacation bounce message.

And, if the email does go through, don't ignore it. Acknowledge it, either agreeing or pushing back on the comments.

If you don't acknowledge the effort I just spent in reviewing your submission, that will make me very unlikely to ever want to review it again in the future.

Don't resend without saying why

If you take my review comments, and resend the patch, and don't say what you did different from the first patch submission, I'll think that you just ignored everything that I said in the past and just delete the patch from my mailbox. Based on the patch load I get, I can't remember what I wrote about your specific patch, so don't assume that I do.

What I will do for you:

So, finally, you created the perfect patch series, took all review into account, and sent it correctly, without corrupting the patch.

What should you expect from me, the subsystem maintainer?

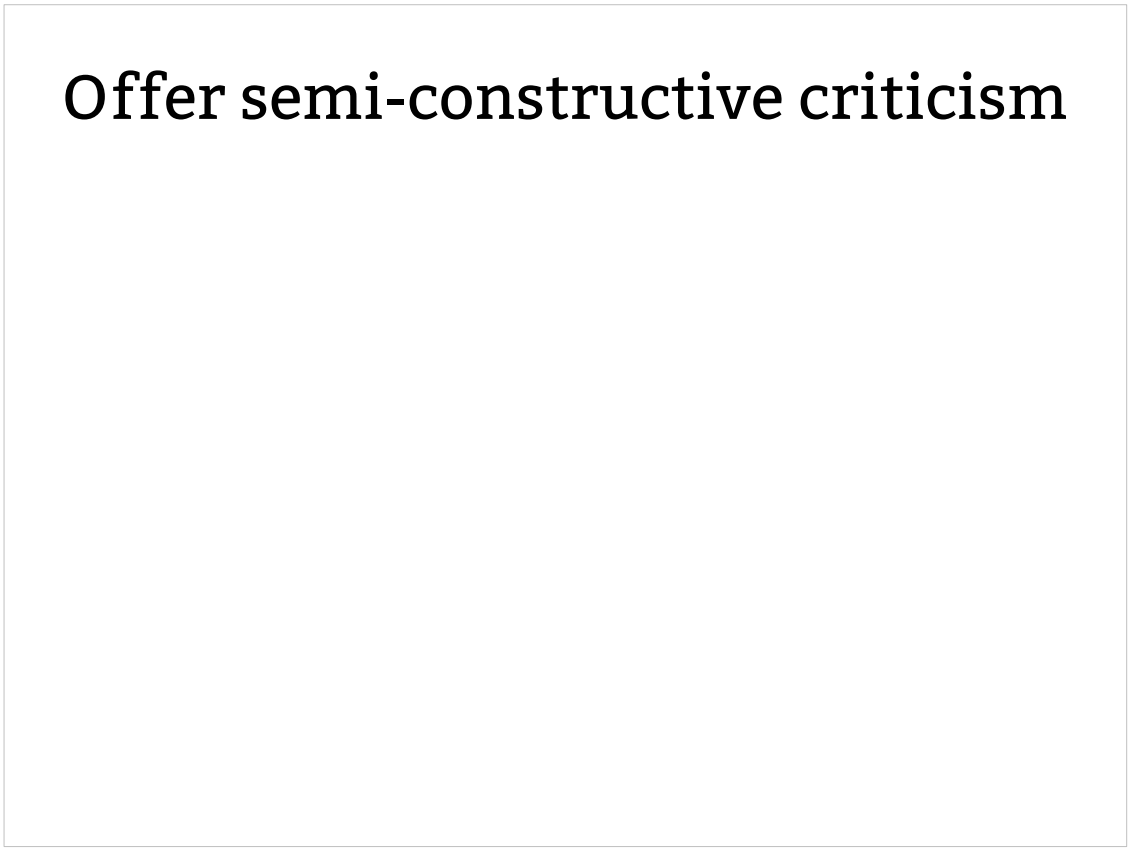
Review your patch within 1-2 weeks

Some subsystem maintainers try to get to patches even faster than this, but with travel and different conferences, the best that I can normally do is about 1-2 weeks.

If I don't respond in that time frame, just ask what is going on. I have no problem with people asking about their patch status. Sometimes patches end up getting dropped on the floor accidentally, and if I'm being slow I have no problem with being called on it, so don't feel bad about checking up on it.

But please wait 1-2 weeks, don't be rude and send a patch at night, and then in the morning send a complaining email asking why it wasn't reviewed already. This happens more than you want to know.

Offer semi-constructive criticism



I can't always promise constructive criticism, but I'll try my best.

Let you know the status of your patch

I have a set of scripts that I got from Andrew Morton that will email you when I apply your patch to one of my development trees saying where it has been applied and when you can expect to see it show up in Linus's tree. There is no reason that all kernel maintainers shouldn't do this, and it's nice to see that more and more are.

But, I know from personal experience, there are maintainers in this room that I send patches to and I never know what happens to them. A few months later I will see them show up in Linus's tree, usually after I forgot about them.

That's not acceptable, and you should not allow this, push back on your subsystem maintainer to use something like this, to keep you informed. Andrew's scripts are public, as are my variations of them, for everyone to use.

> “Publicly making fun of people is half the fun of open source programming.
>
> In fact the main reason to eschew programming in closed environments is that you can't embarrass people in public.”
>
> – Linus Torvalds

Linus said this after I wrote a small rant about some truly horrible Linux kernel driver code that I found online.

It had all sorts of "this code is never to be included in the Linux kernel" warnings all over it, despite it being a Linux kernel driver. And in reading the code, it was obvious why the author never wanted it in the kernel, it was one of the worse things I had ever seen, and that says a lot. After I complained about it, I felt bad, as someone had obviously spent a lot of time on it, but Linus replied with the above quote.

And as always, it turns out that Linus is right.

> “Publicly making fun of people is half the fun of open source programming.
>
> In fact the main reason to eschew programming in closed environments is that you can't embarrass people in public.”
>
> – Linus Torvalds

If that author had ever thought that the code would have been posted publicly, they wouldn't have written such crap. That's what maintainers and public code review is really for in the end, keeping the crap out of the Linux kernel, which benefits everyone involved.

So while it seems that we kernel developers can be a real harsh bunch of people, it is because of that harshness that Linux is as good as it is.

You want us to be tough, because it makes you better programmers in the end, knowing you will have to defend your code.

And that is why I love doing this work, it makes everyone involved produce the best possible code, which in the end, is what matters the most.

Obligatory Penguin Picture