The kernel’s limits to growth

(LPC/OSS 2017)

Jonathan Corbet
LWN.net
corbet@lwn.net


IF IT'S NOT FUN
WHY DO IT?
Crucial

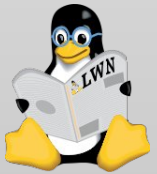
Quite frankly, this particular discussion (and others before it) has just made me irritable, and is ADDING pressure. Instead, I'd suggest that if you have a complaint about how I handle patches, you think about what I end up having to deal with for five minutes.

Go away, people. Or at least don't Cc me any more. I'm not interested, I'm taking a vacation, and I don't want to hear about it any more. In short, get the hell out of my mailbox.
— Linus Torvalds

Quite frankly, this particular discussion (and others before it) has just made me irritable, and is ADDING pressure. Instead, I'd suggest that if you have a complaint about how I handle patches, you think about what I end up having to deal with for five minutes.

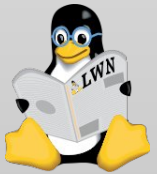
Go away, people. Or at least don't Cc me any more. I'm not interested, I'm taking a vacation, and I don't want to hear about it any more. In short, get the hell out of my mailbox.
— Linus Torvalds, September 1998

Development process scalability



More recently

2.2.0: 1999-01-16
2.4.0: 2001-01-04
2.6.0: 2003-12-17

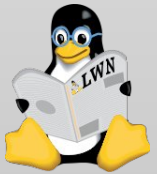


More recently

2.2.0: 1999-01-16
2.4.0: 2001-01-04
2.6.0: 2003-12-17

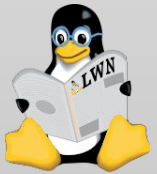
The fun of those days

- Massive backporting of 2.6 patches to 2.4
- Vendor Frankenstein kernels
- Lots of out-of-tree code shipped
- Painful upgrades



So what did we do?

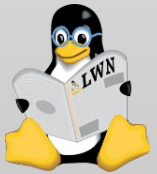
The “upstream first” rule



So what did we do?

The “upstream first” rule

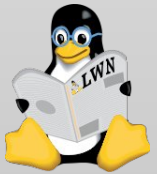
Distributed source-code control



So what did we do?

The “upstream first” rule

Distributed source-code control
(...actually, *any* source-code control...)

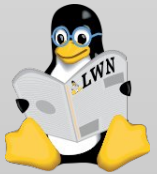


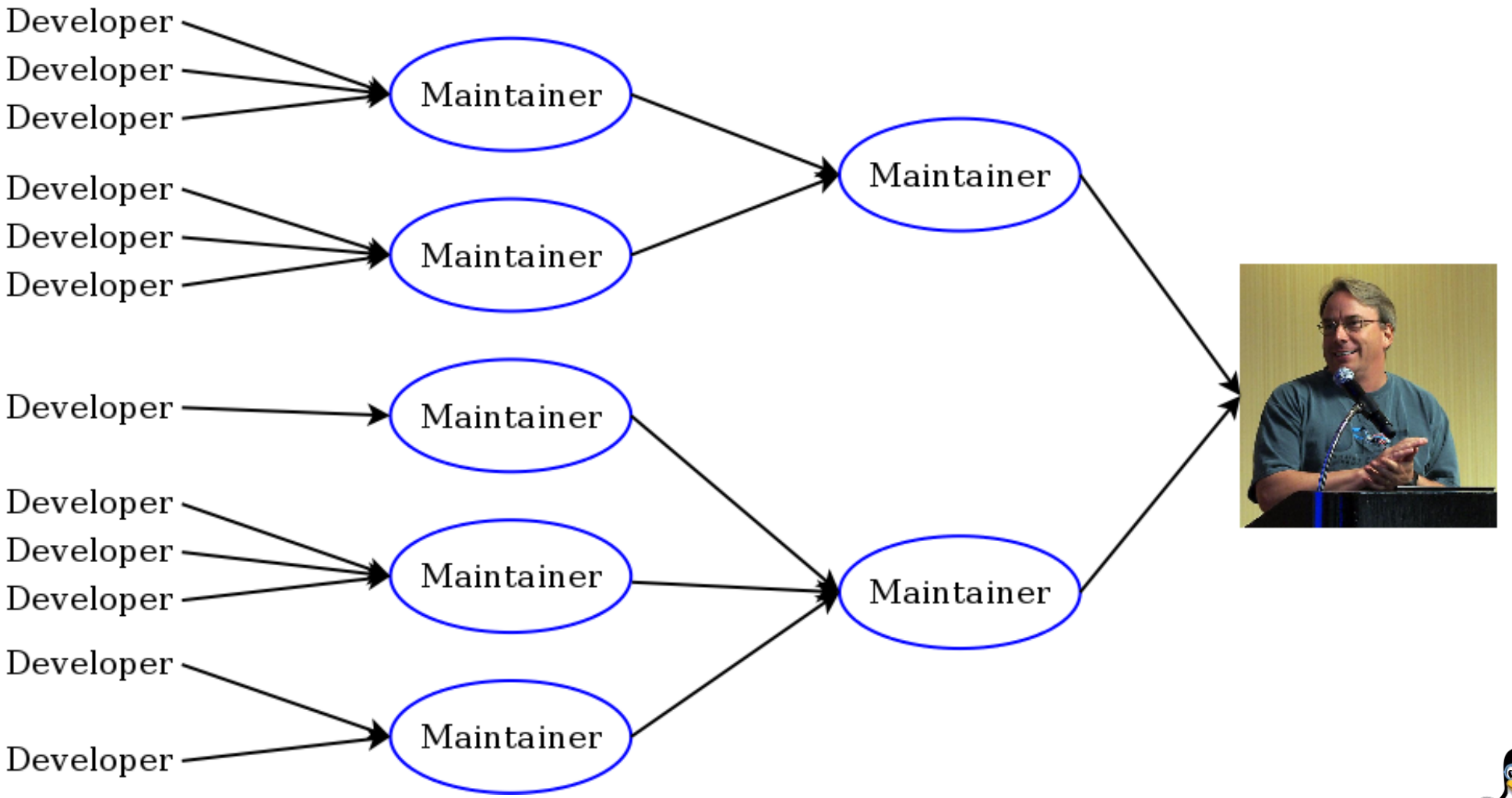
So what did we do?

The “upstream first” rule

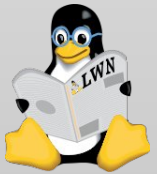
Distributed source-code control
(...actually, *any* source-code control...)

The “new” release model



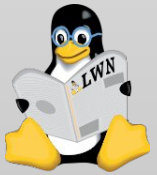

Developer
Developer
Developer
Maintainer
Developer
Developer
Developer
Maintainer
Maintainer
Developer
Developer
Developer
Maintainer
Developer
Developer
Maintainer
Maintainer

So what did those changes do for us?



Recent releases

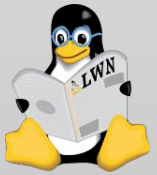
Version	Date	Days	Devs	Changesets
4.7	Jul 17	70	1,582	12,283
4.8	Oct 2	70	1,597	13,382
4.9	Dec 11	70	1,729	16,216
4.10	Feb 19	70	1,672	13,029
4.11	Apr 30	70	1,741	12,724
4.12	Jul 2	63	1,821	14,570
4.13	Sep 3	63	1,681	13,006



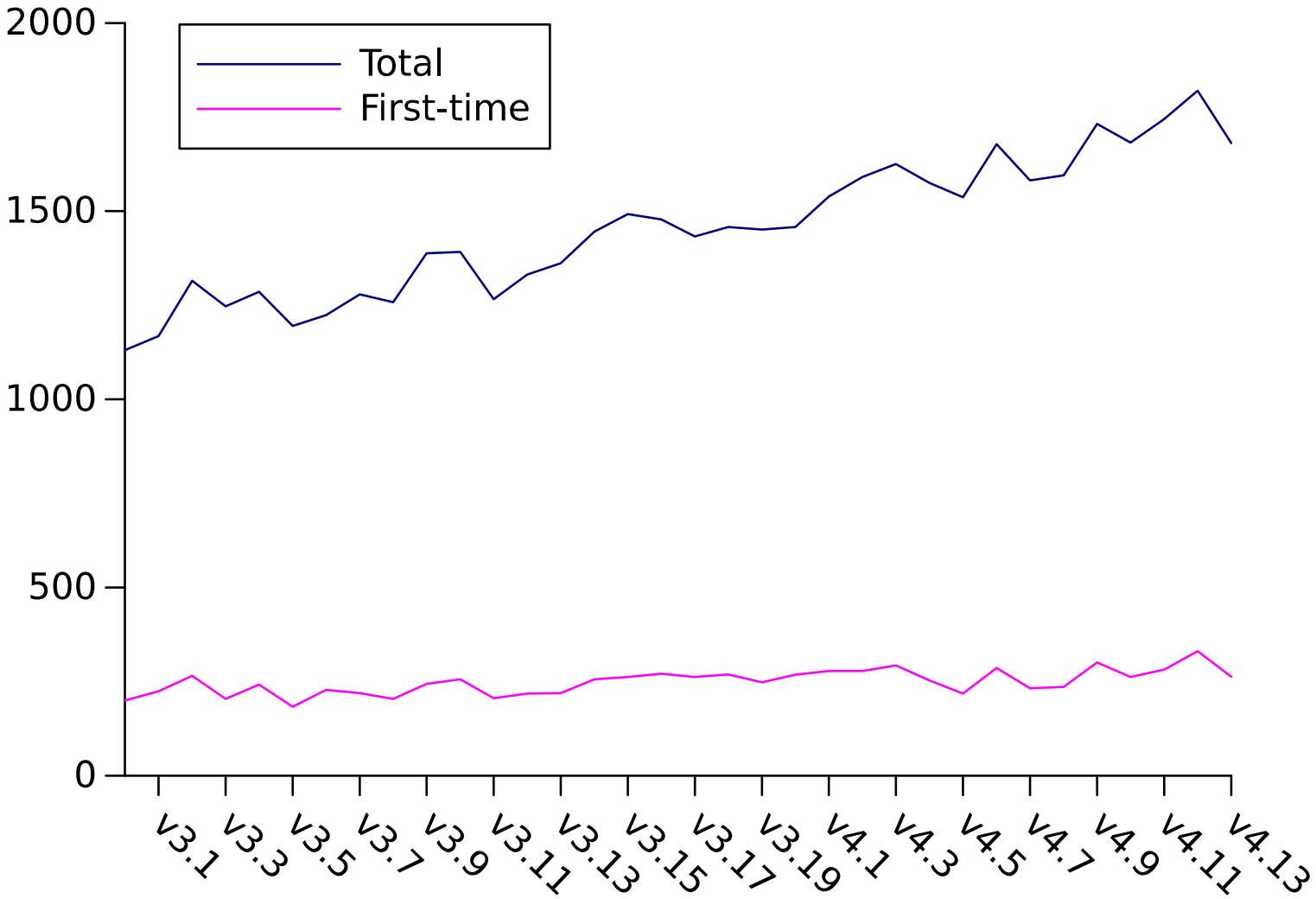
Recent releases

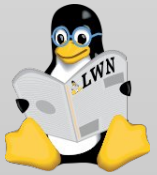
Version	Date	Days	Devs	Changesets
4.7	Jul 17	70	1,582	12,283
4.8	Oct 2	70	1,597	13,382
4.9	Dec 11	70	1,729	16,216
4.10	Feb 19	70	1,672	13,029
4.11	Apr 30	70	1,741	12,724
4.12	Jul 2	63	1,821	14,570
4.13	Sep 3	63	1,681	13,006

→82,925 changes from 4,319 devs since 4.7



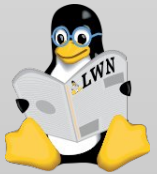
Developers contributing to each release





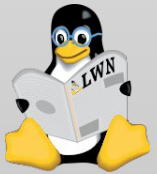
The Linux kernel is everywhere

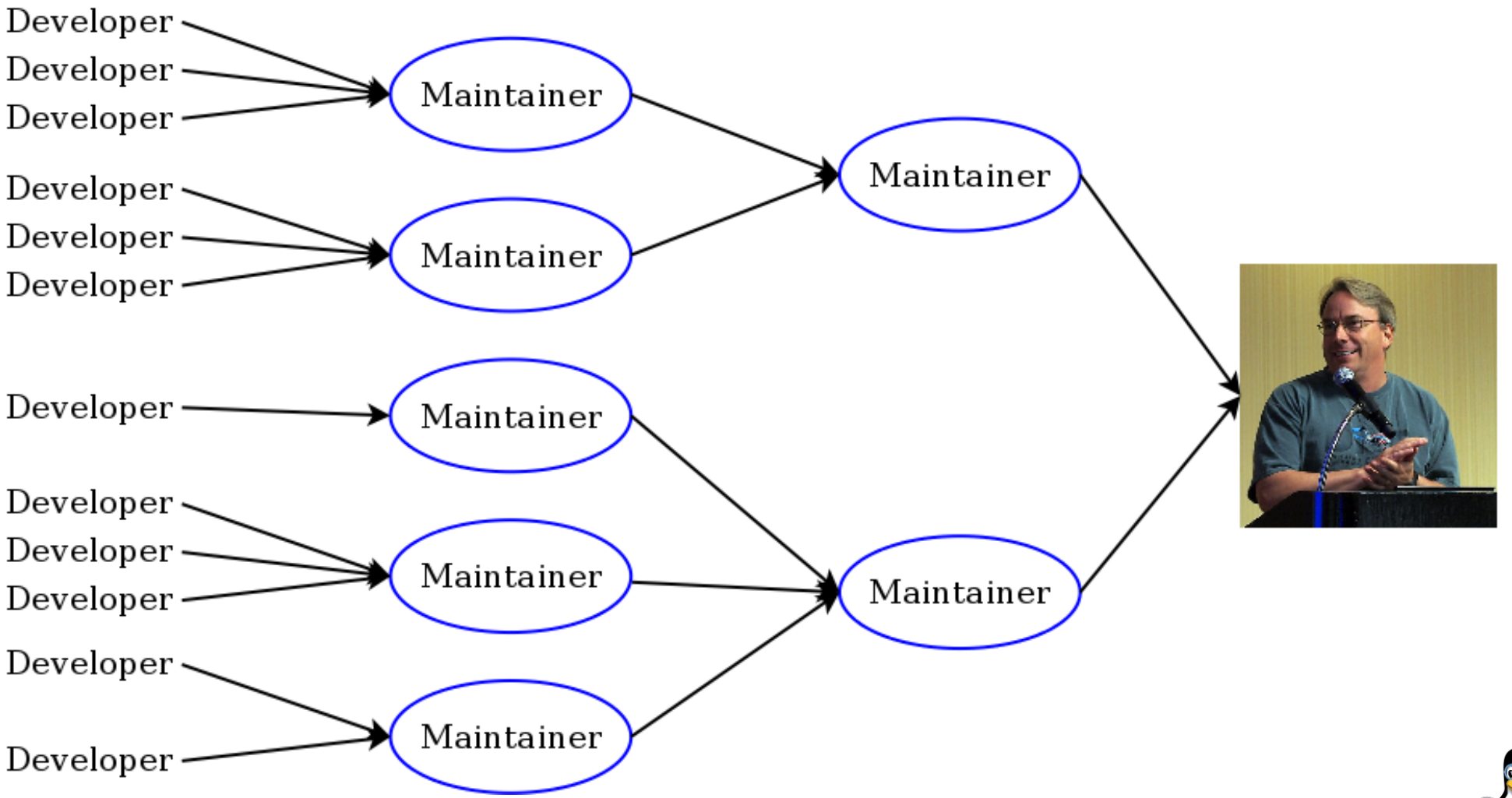
We would appear to be on a roll...

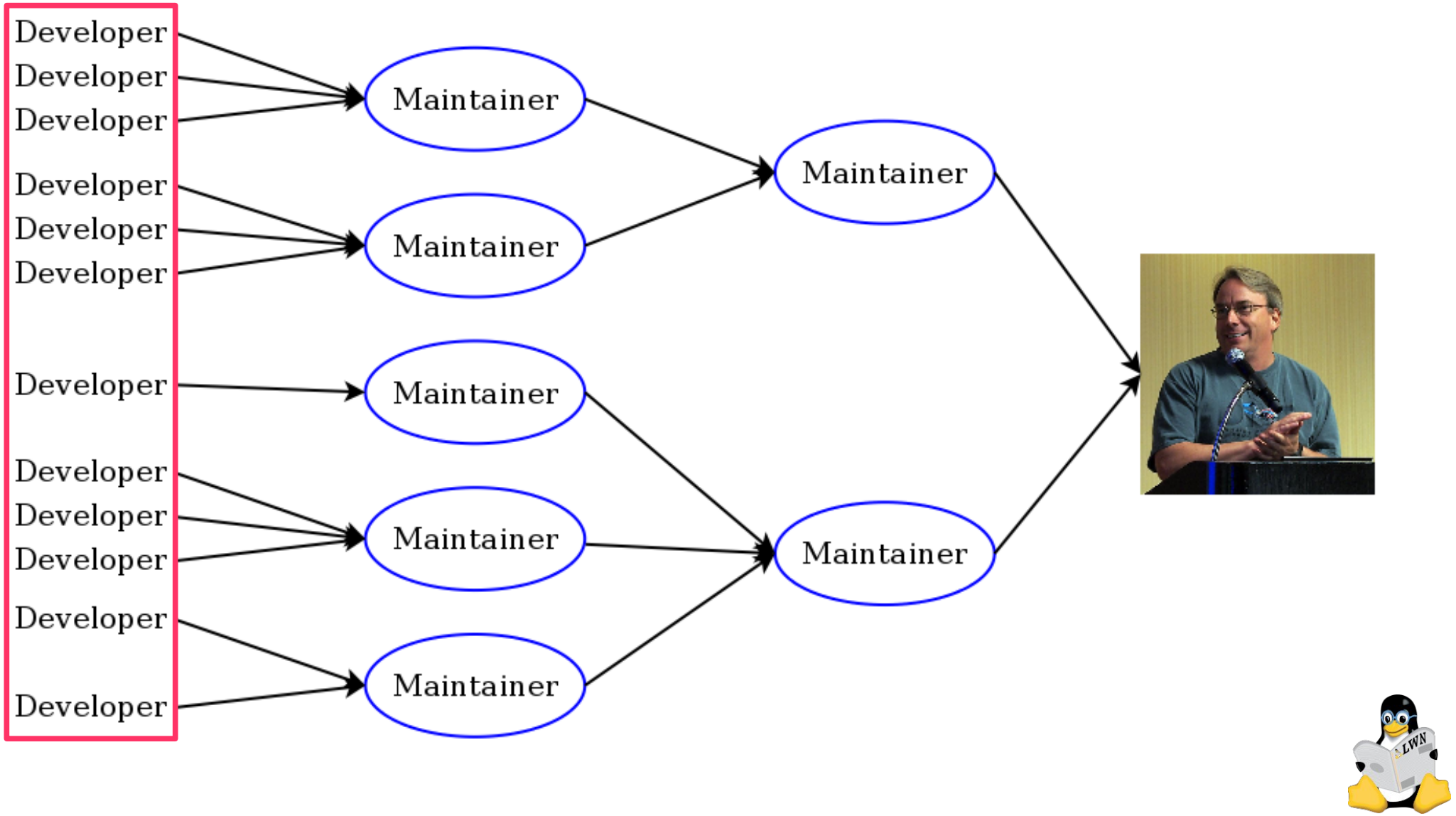


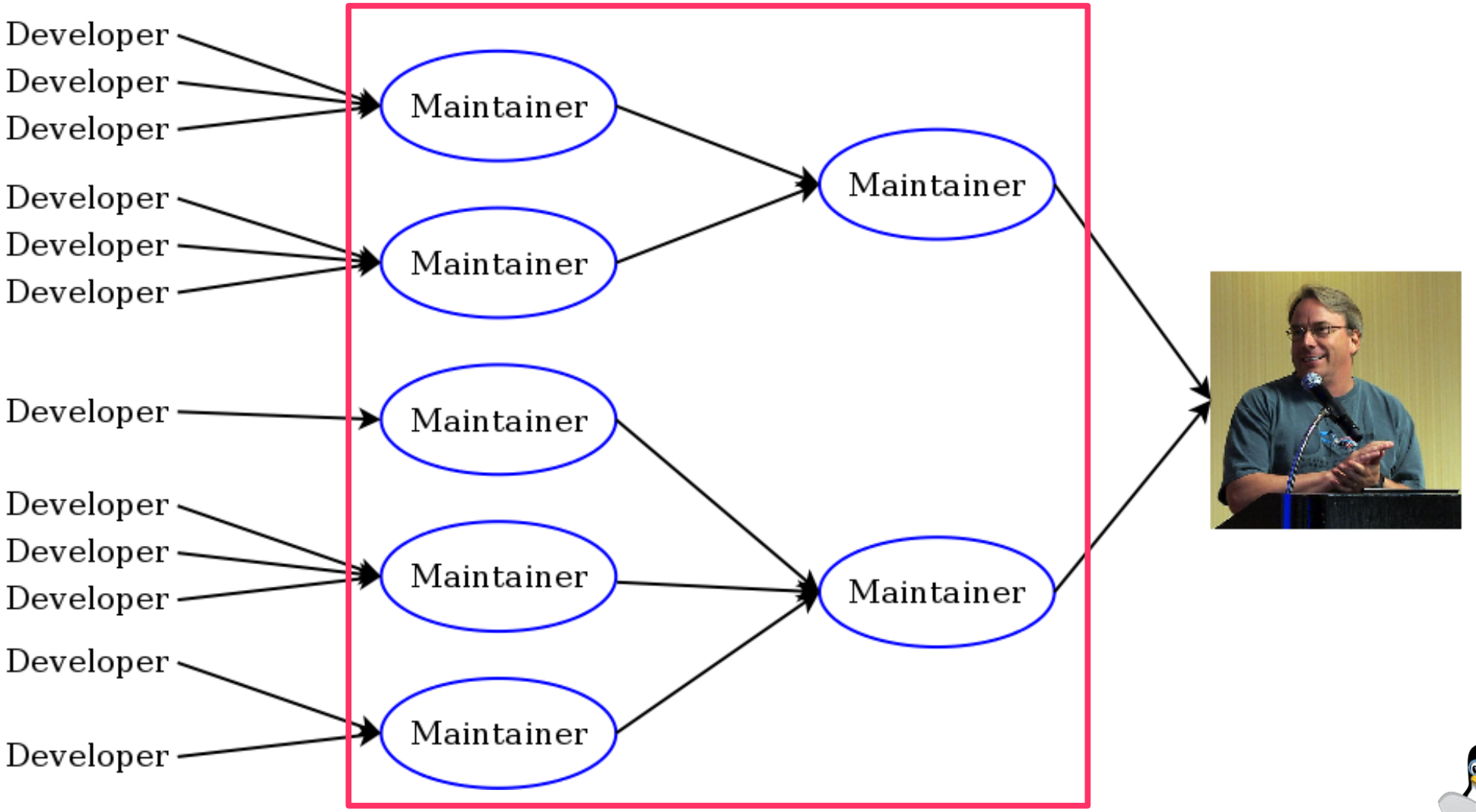
We would appear to be on a roll...

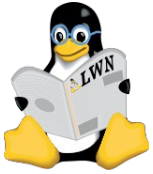
So why am I worried?




Developer
Developer
Developer
Maintainer
Developer
Developer
Developer
Maintainer
Maintainer
Developer
Developer
Developer
Maintainer
Developer
Developer
Maintainer
Maintainer


Developer
Developer
Developer
Developer
Developer
Developer
Developer
Developer
Developer
Developer
Maintainer
Maintainer
Maintainer
Maintainer
Maintainer
Maintainer
LWN


Developer
Developer
Developer
Developer
Developer
Developer
Developer
Developer
Developer
Developer
Developer
Maintainer
Maintainer
Maintainer
Maintainer
Maintainer
Maintainer


LWN

“Roads and bridges”

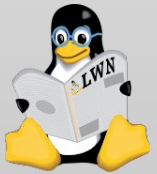
Nadia Eghbal

We are not paying sufficient attention to the needs of our maintainers

Unpaid maintenance?

Maint. support v4.7..

- 21% Red Hat
- 10% Intel
- 9% Linux Foundation
- 8% Linaro
- 7% Google
- 4% Samsung
- 4% —
- 3% IBM
- 2% SUSE



Unpaid maintenance?

Maint. support v4.7..

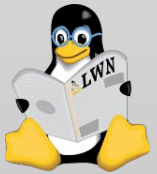
- 21% Red Hat
- 10% Intel
- 9% Linux Foundation
- 8% Linaro
- 7% Google
- 4% Samsung
- 4% —
- 3% IBM
- 2% SUSE

Core maint support

- 30% Google
- 24% Red Hat
- 8% Facebook
- 7% SUSE
- 5% Intel
- 4% consultants
- 4% Huawei
- 4% Linutronix
- 3% Oracle

Work nobody will pay for

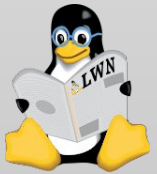
Much core-kernel work
Documentation



I think the problem is a lack of paid tech writers working on Linux. Who would pay them? This lack of documentation makes learning and discovering new Linux features difficult.
— Brendan Gregg, September 2017

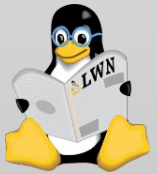
Work nobody will pay for

Much core-kernel work
Documentation
Configuration system
Debloating
Security
...



Security worries

We have no “security officer”
no security training
no security documentation



The year in CVE numbers

CVE-2016-0723 CVE-2016-0728 CVE-2016-0758 CVE-2016-0774 CVE-2016-0821 CVE-2016-0823
CVE-2016-1237 CVE-2016-1575 CVE-2016-1576 CVE-2016-1583 CVE-2016-2053 CVE-2016-2059
CVE-2016-2061 CVE-2016-2062 CVE-2016-2063 CVE-2016-2064 CVE-2016-2065 CVE-2016-2066
CVE-2016-2067 CVE-2016-2068 CVE-2016-2069 CVE-2016-2070 CVE-2016-2085 CVE-2016-2117
CVE-2016-2143 CVE-2016-2184 CVE-2016-2185 CVE-2016-2186 CVE-2016-2187 CVE-2016-2188
CVE-2016-2383 CVE-2016-2384 CVE-2016-2543 CVE-2016-2544 CVE-2016-2545 CVE-2016-2546
CVE-2016-2547 CVE-2016-2548 CVE-2016-2549 CVE-2016-2550 CVE-2016-2782 CVE-2016-2847
CVE-2016-2853 CVE-2016-2854 CVE-2016-3070 CVE-2016-3134 CVE-2016-3135 CVE-2016-3136
CVE-2016-3137 CVE-2016-3138 CVE-2016-3139 CVE-2016-3140 CVE-2016-3156 CVE-2016-3157
CVE-2016-3672 CVE-2016-3689 CVE-2016-3707 CVE-2016-3713 CVE-2016-3841 CVE-2016-3951
CVE-2016-3955 CVE-2016-3961 CVE-2016-4440 CVE-2016-4470 CVE-2016-4482 CVE-2016-4485
CVE-2016-4486 CVE-2016-4557 CVE-2016-4558 CVE-2016-4565 CVE-2016-4568 CVE-2016-4569
CVE-2016-4578 CVE-2016-4580 CVE-2016-4581 CVE-2016-4794 CVE-2016-4805 CVE-2016-4913
CVE-2016-4951 CVE-2016-4997 CVE-2016-4998 CVE-2016-5243 CVE-2016-5244 CVE-2016-5340
CVE-2016-5342 CVE-2016-5344 CVE-2016-5400 CVE-2016-5412 CVE-2016-5696 CVE-2016-5728
CVE-2016-5828 CVE-2016-5829 CVE-2016-6130 CVE-2016-6136 CVE-2016-6156 CVE-2016-6162
CVE-2016-6187 CVE-2016-6197 CVE-2016-6198 CVE-2016-6480 [...]

```
diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
index c701f6a..5c16e24 100644
--- a/net/ipv4/tcp_input.c
+++ b/net/ipv4/tcp_input.c
@@ -4420,9 +4420,11 @@ int tcp_rcv_state_process(struct sock *sk, struct sk_buff *skb,
                         * But, this leaves one open to an easy denial of
                         * service attack, and SYN cookies can't defend
                         * against this problem. So, we drop the data
-                        * in the interest of security over speed.
+                        * in the interest of security over speed unless
+                        * it's still in use.
                         */
-                       goto discard;
+                       kfree_skb(skb);
+                       return 0;
                }
```

```
diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
index c701f6a..5c16e24 100644
--- a/net/ipv4/tcp_input.c
+++ b/net/ipv4/tcp_input.c
@@ -4420,9 +4420,11 @@ int tcp_rcv_state_process(struct sock *sk, struct sk_buff *skb,
                          * But, this leaves one open to an easy denial of
                          * service attack, and SYN cookies can't defend
                          * against this problem. So, we drop the data
-                         * in the interest of security over speed.
+                         * in the interest of security over speed unless
+                         * it's still in use.
                          */
-                        goto discard;
+                        kfree_skb(skb);
+                        return 0;
                }
```

2007

2007

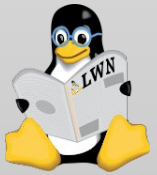
```
diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
index c701f6a..5c16e24 100644
--- a/net/ipv4/tcp_input.c
+++ b/net/ipv4/tcp_input.c
@@ -4420,9 +4420,11 @@ int tcp_rcv_state_process(struct sock *sk, struct sk_buff *skb,
                         * But, this leaves one open to an easy denial of
                         * service attack, and SYN cookies can't defend
                         * against this problem. So, we drop the data
-                        * in the interest of security over speed.
+                        * in the interest of security over speed unless
+                        * it's still in use.
                         */
-                       goto discard;
+                       kfree_skb(skb);
+                       return 0;
                }
```

```
diff --git a/net/dccp/input.c b/net/dccp/input.c
index ba34718..8fedc2d 100644
--- a/net/dccp/input.c
+++ b/net/dccp/input.c
@@ -606,7 +606,8 @@ int dccp_rcv_state_process(struct sock *sk, struct sk_buff *skb,
                        if (inet_csk(sk)->icsk_af_ops->conn_request(sk,
                                                                    skb) < 0)
                                return 1;
-                       goto discard;
+                       consume_skb(skb);
+                       return 0;
                }
                if (dh->dccph_type == DCCP_PKT_RESET)
                        goto discard;
```

2007

```
diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
index c701f6a..5c16e24 100644
--- a/net/ipv4/tcp_input.c
+++ b/net/ipv4/tcp_input.c
@@ -4420,9 +4420,11 @@ int tcp_rcv_state_process(struct sock *sk, struct sk_buff *skb,
                         * But, this leaves one open to an easy denial of
                         * service attack, and SYN cookies can't defend
                         * against this problem. So, we drop the data
-                        * in the interest of security over speed.
+                        * in the interest of security over speed unless
+                        * it's still in use.
                         */
-                       goto discard;
+                       kfree_skb(skb);
+                       return 0;
                }
```

```
diff --git a/net/dccp/input.c b/net/dccp/input.c
index ba34718..8fedc2d 100644
--- a/net/dccp/input.c
+++ b/net/dccp/input.c
@@ -606,7 +606,8 @@ int dccp_rcv_state_process(struct sock *sk, struct sk_buff *skb,
                        if (inet_csk(sk)->icsk_af_ops->conn_request(sk,
                                                                    skb) < 0)
                                return 1;
-                       goto discard;
+                       consume_skb(skb);
+                       return 0;
                }
                if (dh->dccph_type == DCCP_PKT_RESET)
                        goto discard;
```



```
diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
index c701f6a..5c16e24 100644
--- a/net/ipv4/tcp_input.c
+++ b/net/ipv4/tcp_input.c
@@ -4420,9 +4420,11 @@ int tcp_rcv_state_process(struct sock *sk, struct sk_buff *skb,
                         * But, this leaves one open to an easy denial of
                         * service attack, and SYN cookies can't defend
                         * against this problem. So, we drop the data
-                        * in the interest of security over speed.
+                        * in the interest of security over speed unless
+                        * it's still in use.
                         */
-                       goto discard;
+                       kfree_skb(skb);
+                       return 0;
                }
```

2007

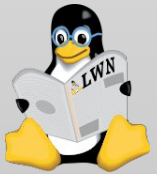
```
diff --git a/net/dccp/input.c b/net/dccp/input.c
index ba34718..8fedc2d 100644
--- a/net/dccp/input.c
+++ b/net/dccp/input.c
@@ -606,7 +606,8 @@ int dccp_rcv_state_process(struct sock *sk, struct sk_buff *skb,
                        if (inet_csk(sk)->icsk_af_ops->conn_request(sk,
                                                        skb) < 0)
                                return 1;
-                       goto discard;
+                       consume_skb(skb);
+                       return 0;
                }
                if (dh->dccph_type == DCCP_PKT_RESET)
                        goto discard;
```

CVE-2017-6074

Security shows a big hole in our maintainer model

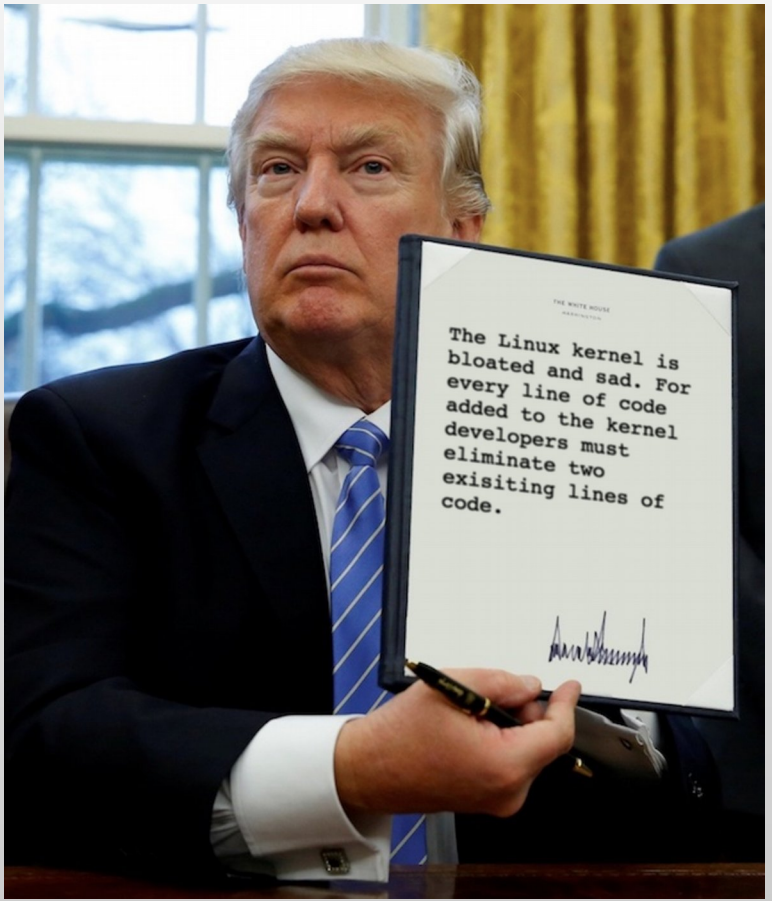
Unmaintained code

There are some dark corners in the kernel tree.

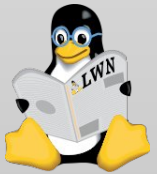


Unmaintained code

There are some dark corners in the kernel tree.

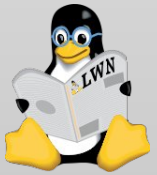


Some other concerns



Review bandwidth

> The big problem is this, we really only have a very small group of people reviewing code in the kernel community.
> — Greg Kroah-Hartman



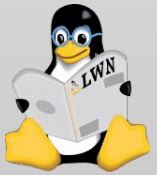
Review bandwidth

> The big problem is this, we really only have a very small group of people reviewing code in the kernel community.
> — Greg Kroah-Hartman, 2006

Review bandwidth

> The big problem is this, we really only have a very small group of people reviewing code in the kernel community.
> — Greg Kroah-Hartman, 2006

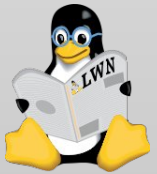
> I am worried that the number of patches posted to linux-mm grows over time while the number of reviewers doesn’t scale up with that trend.
> — Michal Hocko, 2017



Wolfram Sang: the number of reviewers is not scaling with the number of contributors.

As a consequence

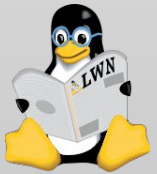
Maintainers burn out and fall behind



As a consequence

Maintainers burn out and fall behind

Unreviewed code gets in

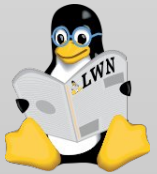


As a consequence

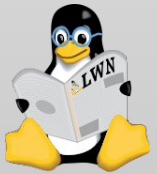
Maintainers burn out and fall behind

Unreviewed code gets in

Long-term API problems



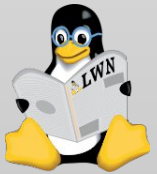
Review bandwidth is a problem for all projects



We work hard to encourage contributions

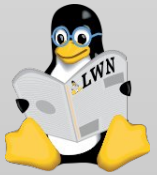
Perhaps we should do more to promote code-review skills?

Out-of-tree code



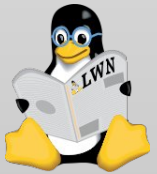
Mobile SoC code out-of-tree

Company	Phone	SOC	Insertions
LG	G3	Msm	2.6 M
Motorola	Moto X	Msm	1.8 M
Samsung	Galaxy 4	Exynos	1.1 M
Samsung	Galaxy S5	Msm	3.1 M
Sony	Xperia Z2	Msm	1.8 M
Sony	Xperia C	Mediatek	1.9 M
Acer	Liquid E2	Mediatek	1.4 M
Asus	Zenfone 6	Atom	2.2 M
Huawei	Ascend P7	Hisilicon	2.7 M

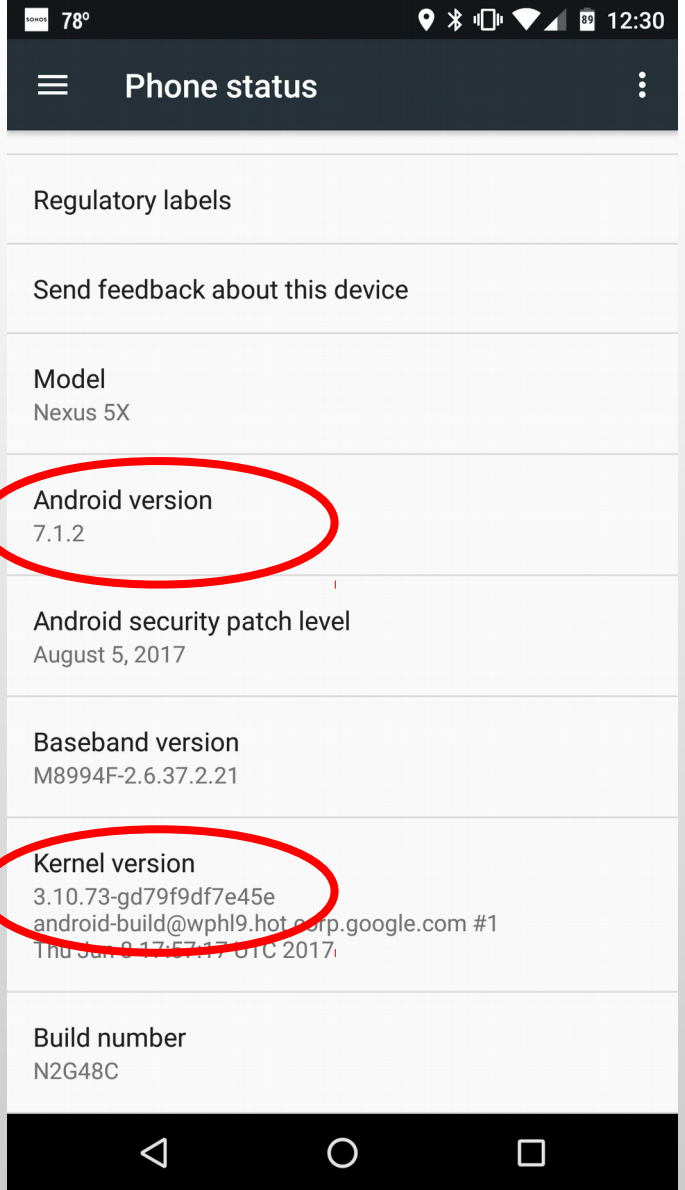


Out-of-tree code consequences

Bugs and security issues
Inability to run mainline kernels
Maintainer stress
Maintainers pulled out of the community



My phone


78°
12:30
Phone status
Regulatory labels
Send feedback about this device
Model
Nexus 5X
Android version
7.1.2
Android security patch level
August 5, 2017
Baseband version
M8994F-2.6.37.2.21
Kernel version
3.10.73-gd79f9df7e45e
android-build@wphl9.hot.corp.google.com #1
Thu Jan 8 17:57:17 UTC 2017
Build number
N2G48C


LWN

The 3.10 kernel

Was released in June 2013
- 3.10.73 update was March 2015

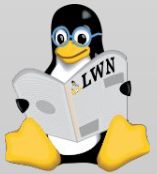
Is 300,000 patches behind the mainline

More recently

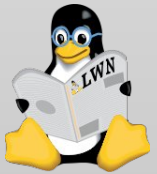
2.2.0: 1999-01-16
2.4.0: 2001-01-04
2.6.0: 2003-12-17

The fun of those days

- Massive backporting of 2.6 patches to 2.4
- Vendor Frankenstein kernels
- Lots of out-of-tree code shipped
- Painful upgrades



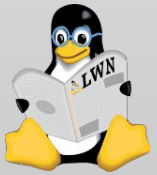
How much more can we grow with this much energy being directed away from our community?



What is maintainership?

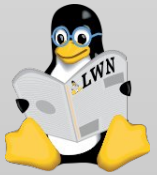
How does one become a maintainer?

Maintainers tend to get to be maintainers because they were good at something else, and not good enough at hiding from the "maintainer" role. There is a paradox here as a maintainer must be good at saying "No", but if they were they might never have agreed to become a maintainer.
— Neil Brown



How does one stop?

> I’m trying to appear to be an incompetent maintainer so that someone will offer to take over. It isn’t working yet.
> — Neil Brown



How does one stop?

I'm trying to appear to be an incompetent maintainer so that someone will offer to take over. It isn't working yet.
— Neil Brown

I have decided to fall back on the mechanism by which I ended up being maintainer in the first place. I will create a vacuum and hope somebody fills it.
— Neil Brown

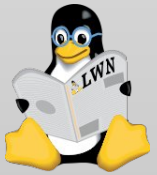
What is a maintainer’s authority?

You should always be able to handle other people changing files in your area at any point in time. Kernel maintainership is not “no one else can ever touch this!” type of development.
— Greg Kroah-Hartman

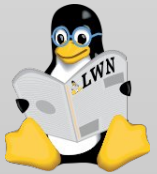
What is a maintainer’s authority?

You should always be able to handle other people changing files in your area at any point in time. Kernel maintainership is not “no one else can ever touch this!” type of development.
— Greg Kroah-Hartman

It is *my* prerogative to say no to anything in arch/arm, and I really don’t have to give reasons for it if I choose to.
— Russell King

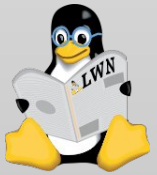


“A bunch of little fiefdoms”



What are a maintainer’s responsibilities?

> I can’t take patches without a changelog text, and neither should any other maintainer.
> — Greg Kroah-Hartman



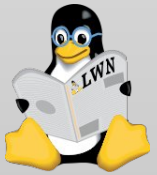
What are a maintainer’s responsibilities?

> I can’t take patches without a changelog text, and neither should any other maintainer.
> — Greg Kroah-Hartman

(536 no-changelog patches were merged for 4.10)

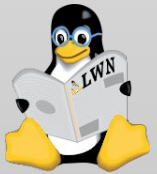
What are a maintainer’s responsibilities?

Review the code
Mentor developers
Respond quickly to patches
Check code provenance
Respond to regressions
Route fixes to -stable
Represent the subsystem to the world
Resist company pressure
Keep Linus happy
[...]



Patch management

Not dropping patches through the cracks
Proper Git repository practices
Informing contributors about actions
Avoiding / handling conflicts
...



Why kernel development still uses email

By **Jonathan Corbet**
October 1, 2016

Kernel Recipes

In a world full of fancy development tools and sites, the kernel project's dependence on email and mailing lists can seem quaintly dated, if not positively prehistoric. But, as Greg Kroah-Hartman pointed out in a Kernel Recipes talk titled "Patches carved into stone tablets", there are some good reasons for the kernel community's choices. Rather than being a holdover from an older era, email remains the best way to manage a project as large as the kernel.

In short, Greg said, kernel developers still use email because it is faster than any of the alternatives. Over the course of the last year, the project accepted about eight changes per hour — every hour — from over 4,000 developers sponsored by over 400 companies. It must be doing something right. The list of maintainers who accepted at least one patch per day contains 75 entries; at the top of the list, Greg himself accepted 9,781 patches over the year. Given that he accepts maybe one third of the patches sent his way, it is clear that the patch posting rate is much higher than that.

Finding tools that can manage that sort of patch rate is hard. A poor craftsman famously complains about his tools, Greg said, but a good craftsman knows how to choose excellent tools.

So which tools are available for development work? Greg started by looking at GitHub, which, he said, has a number of advantages. It is "very very pretty" and is easy to use for small projects thanks to its simple interface. GitHub offers free hosting and unlimited bandwidth, and can (for a fee) be run on a company's own infrastructure. It makes life easy for the authors of drive-by patches; Greg uses it for the usbutils project and gets an occasional patch that way.

On the other hand, GitHub does not scale to larger projects. He pointed at the Kubernetes project, which has over 4,000 open issues and 511 open pull requests. The system, he said, does not work well for large numbers of reviewers. It has a reasonable mechanism for discussion threads attached to pull requests — GitHub has duplicated email for that feature, he said — but only the people who are actually assigned to a pull request can see that thread. GitHub also requires online access, but there are a lot of kernel developers who, for whatever reason, do not have good access to the net while they are working. In general, it is getting better, but projects like Kubernetes are realizing that they need to find something better suited to their scale; it would never work for the kernel.

Moving on to Gerrit, Greg started to list its good points, but stopped short, saying he didn't know any. Actually, there was one: project managers love it, since it gives them the feeling that they know what is going on within the project. He noted that Google, which promotes Gerrit for use with the Android project, does not use it for any of its internal projects. Even with Android, Gerrit is not really needed; Greg pointed out that, in the complicated flow chart showing how to get a patch into Android, Gerrit has a small and replaceable role.

Gerrit, he said, makes patch submission quite hard; Repo helps a bit in that regard, but not many projects use it. Gerrit can be scripted, but few people do that. An audience member jumped in to say that using Gerrit was like doing one's

Why kernel development still uses email

By **Jonathan Corbet**
October 1, 2016

Kernel Recipes

In a world full of fancy development tools and sites, the kernel project's dependence on email and mailing lists can seem quaintly dated, if not positively prehistoric. But, as Greg Kroah-Hartman pointed out in a Kernel Recipes talk titled "Patches carved into stone tablets", there are some good reasons for the kernel community's choices. Rather than being a holdover from an older era, email remains the best way to manage a project as large as the kernel.

In short, Greg said, kernel developers still use email because it is faster than any of the
accepted about eight changes per hour — every hour — from over 4,000 developers sp
right. The list of maintainers who accepted at least one patch per day contains 75 entri
over the year. Given that he accepts maybe one third of the patches sent his way, it is

Finding tools that can manage that sort of patch rate is hard. A poor craftsman famousl
knows how to choose excellent tools.

So which tools are available for development work? Greg started by looking at GitHub,
pretty" and is easy to use for small projects thanks to its simple interface. GitHub offers
on a company's own infrastructure. It makes life easy for the authors of drive-by patche
patch that way.

On the other hand, GitHub does not scale to larger projects. He pointed at the Kuberne
requests. The system, he said, does not work well for large numbers of reviewers. It ha
discussion threads attached to pull requests — GitHub has duplicated email for that fea
people who are actually assigned to a pull request can see that thread. GitHub also rec
are a lot of kernel developers who, for whatever reason, do not have good access to th
general, it is getting better, but projects like Kubernetes are realizing that they need to
their scale; it would never work for the kernel.

Moving on to Gerrit, Greg started to list its good points, but stopped short, saying he di
one: project managers love it, since it gives them the feeling that they know what is goi
that Google, which promotes Gerrit for use with the Android project, does not use it for
with Android, Gerrit is not really needed; Greg pointed out that, in the complicated flow
into Android, Gerrit has a small and replaceable role.

Gerrit, he said, makes patch submission quite hard; Repo helps a bit in that regard, but
can be scripted, but few people do that. An audience member jumped in to say that usi

Why Github can't host the Linux Kernel Community

August 8, 2017 • danvet

A while back at the awesome maintainerati I chatted with a few great fellow maintainers about how to scale really big open source projects, and how github forces projects into a certain way of scaling. The linux kernel has an entirely different model, which maintainers hosting their projects on github don't understand, and I think it's worth explaining why and how it works, and how it's different.

Another motivation to finally get around to typing this all up is the HN discussion on my "Maintainers Don't Scale" talk, where the top comment boils down to "… why don't these dinosaurs use modern dev tooling?". A few top kernel maintainers vigorously defend mailing lists and patch submissions over something like github pull requests, but at least some folks from the graphics subsystem would love more modern tooling which would be much easier to script. The problem is that github doesn't support the way the linux kernel scales out to a huge number of contributors, and therefore we can't simply move, not even just a few subsystems. And this isn't about just hosting the git data, that part obviously works, but how pull requests, issues and forks work on github.

Scaling, the Github Way

Git is awesome, because everyone can fork and create branches and hack on the code very easily

Speaking of patch management

Photo: Lars Plougmann

Kids these days do things differently.

Our maintainers are getting older

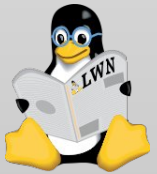
Back to the point

We don’t define the maintainer role well
We don’t document how to fill it
We don’t train future maintainers

Back to the point

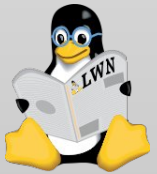
We don’t define the maintainer role well
We don’t document how to fill it
We don’t train future maintainers

How much more can we scale in this mode?

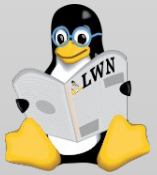


So what can we do?

Recognize maintainership as an activity needing support



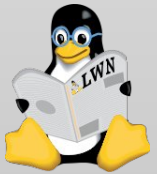
Document what it means to be a maintainer



Create training and mentoring for new maintainers

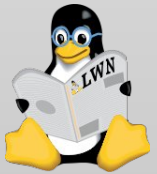
Teach code-review skills and encourage their use

Move away from the single-maintainer model (explore group maintainership)

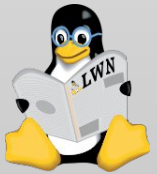


Think about our next generation of tools

Pay more attention to our unmaintained dark corners



Don’t assume our process-scalability problems are behind us



Thank you