Automation in the Maintenance of the Linux Kernel
The Coccinelle Experience

Julia Lawall, Gilles Muller (Inria/LIP6)

February 6, 2019

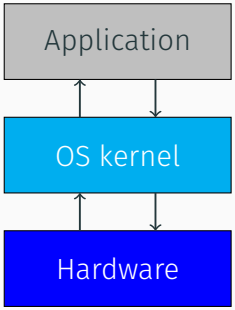
What is an operating system kernel?

What is an operating system kernel?

Contains:

- screen
- keyboard
- camera
- CPU
- disk
- etc.

What is an operating system kernel?



Contains:

- screen
- keyboard
- camera
- CPU
- disk
- etc.

The OS kernel is the software that manages application access to hardware

Operating system correctness is critical

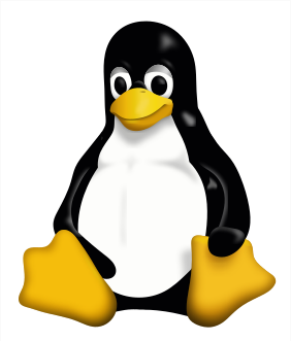
- Our “modern” society (almost) entirely relies on computers
- Applications are directly impacted by any bug in the operating system
  - Application unavailability
  - Loss of data
  - Security attacks
  - Risks for human life
  - Impossible to predict the next exploit

Meltdown

Spectre

Our focus: The Linux kernel

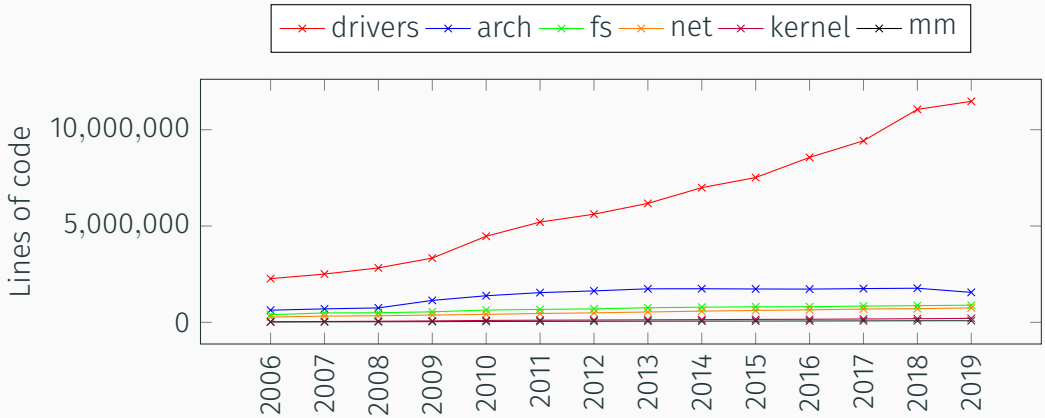
- Open source OS kernel, developed by Linus Torvalds
- First released in 1991
- Version 1.0.0 released in 1994
- Today used in the top 500 supercomputers,
  billions of smartphones (Android),
  battleships,
  stock exchanges, ...



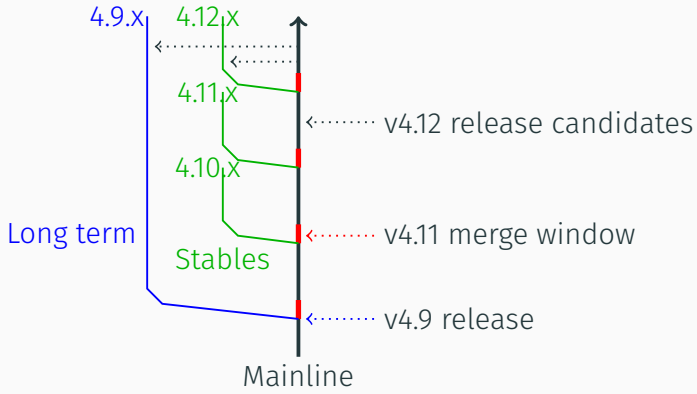
Development challenges

- Large code size
  - 17.5 million lines of code in Linux v4.20 (Dec 2018)
- Multiple streams of development
  - Mainline version, accepting new features and bug fixes.
  - Stable versions, accepting only bug fixes from the mainline.
- Wide range of contributors
  - Almost 20 000 contributors since 2005.
  - Industry developers, hobbyists, newbies.
- High rate of change and heavy code review burden
  - 13-14K commits (changes) per release (every 2-3 months).
  - Need to be integrated with both mainline and stable versions.

Code size

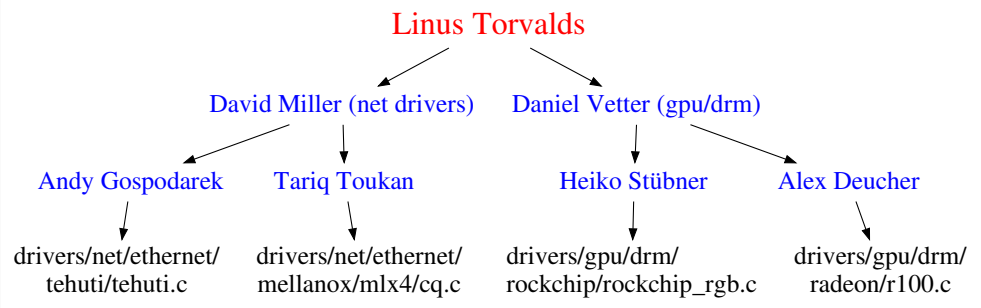

drivers
arch
fs
net
kernel
mm
Lines of code
10,000,000
5,000,000
0
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019

Release model



A new mainline release every 2-3 months.
13-14K commits (changes) per release

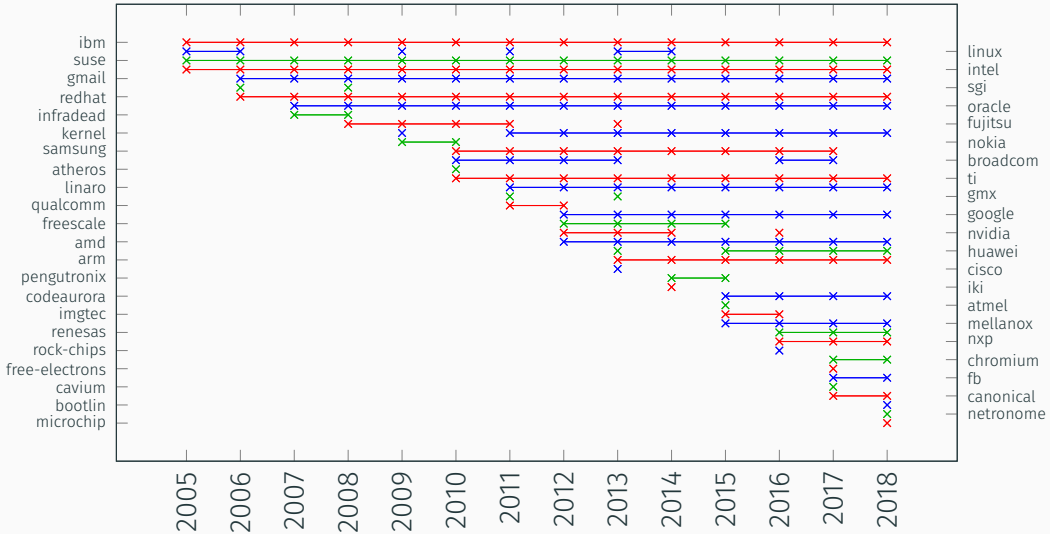
Maintainer hierarchy (tiny extract)



1287 maintainers in v4.20 (Dec 2018), each responsible for 1-8000 files.

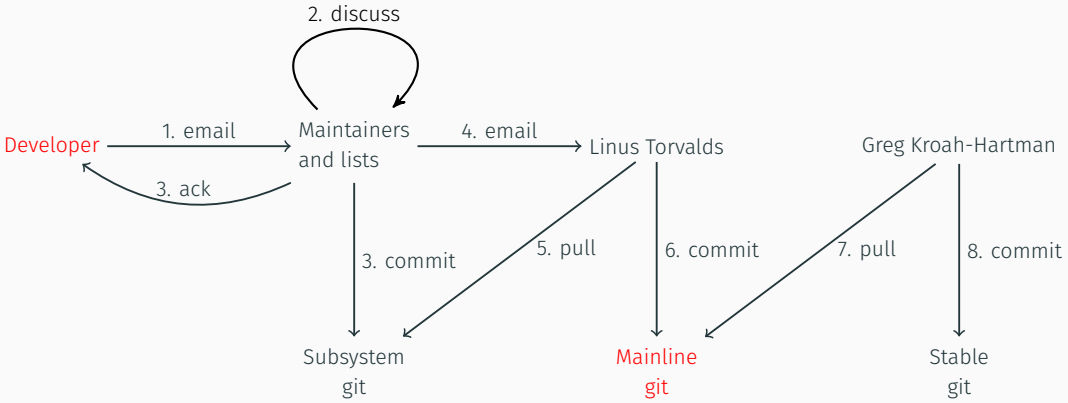
Collaborative development

Largest contributors: $\geq$ 10 developers and $\geq$ 500 commits per year.



Contribution workflow

All contributions go through email



Emailed patch, fixing a bug in a drm driver

```
From nobody Sun Jan 13 09:05:41 CET 2019
From: Julia Lawall <Julia.Lawall@lip6.fr>
To: Sandy Huang <hjc@rock-chips.com>
Cc: "Heiko Stübner" <heiko@sntech.de>,David Airlie <airlied@linux.ie>,Daniel Vetter <daniel@ffwll.ch>,
    dri-devel@lists.freedesktop.org,linux-arm-kernel@lists.infradead.org,linux-rockchip@lists.infradead.org,
    linux-kernel@vger.kernel.org
Subject: [PATCH 3/4] drm/rockchip: add missing of_node_put

The device node iterators perform an of_node_get on each iteration, so a
jump out of the loop requires an of_node_put.
...

Signed-off-by: Julia Lawall <Julia.Lawall@lip6.fr>
---
 drivers/gpu/drm/rockchip/rockchip_rgb.c |    4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)

diff --git a/drivers/gpu/drm/rockchip/rockchip_rgb.c b/drivers/gpu/drm/rockchip/rockchip_rgb.c
index 96ac145..37f9302 100644
--- a/drivers/gpu/drm/rockchip/rockchip_rgb.c
+++ b/drivers/gpu/drm/rockchip/rockchip_rgb.c
@@ -116,2 +116,4 @@ struct rockchip_rgb *rockchip_rgb_init(struct device *dev,
-               if (!ret)
+               if (!ret) {
+                       of_node_put(endpoint);
                        break;
+               }
```

Problem: Big code implies the need for big changes

Thomas Gleixner (maintainer of x86 architecture support, interrupt support, etc.):
Remove the irq argument from interrupt handlers.

- 188 affected files.

Kees Cook (organizer of the Linux kernel self-protection project):
Replace malloc-based array allocations by safer versions.

- 377 and 484 affected files.

Deepa Dinamani (developer), and many others: Time data size for Y2038.

- $\sim$ 250 affected files.

Julia Lawall: Add missing `of_node_put`.

- 50 affected files.

Some patches fixing missing of_node_puts

drivers/video/backlight/88pm860x_bl.c

```
@@ -180,4 +180,5 @@
  data->iset = PM8606_WLED_CURRENT(iset);
  of_property_read_u32(np, "marvell,88pm860x-pwm",
                       &data->pwm);
+ of_node_put(np);
  break;
```

drivers/power/charger-manager.c

```
@@ -1581,8 +1581,10 @@
  cables = devm_kzalloc(dev, sizeof(*cables)
                  * chg_regs->num_cables,
                  GFP_KERNEL);
- if (!cables)
+ if (!cables) {
+       of_node_put(child);
        return ERR_PTR(-ENOMEM);
+ }
```

arch/arm/kernel/devtree.c

```
@@ -101,6 +101,7 @@
  if (of_property_read_u32(cpu, "reg", &hwid)) {
        pr_debug(" * %s missing reg property\n",
                     cpu->full_name);
+       of_node_put(cpu);
        return;
  }

@@ -108,8 +109,10 @@
   * 8 MSBs must be set to 0 in the DT since the reg p
   * defines the MPIDR[23:0].
   */
- if (hwid & ~MPIDR_HWID_BITMASK)
+ if (hwid & ~MPIDR_HWID_BITMASK) {
+       of_node_put(cpu);
        return;
+ }

 /*
  * Duplicate MPIDRs are a recipe for disaster.
```

Some patches fixing missing of_node_puts

drivers/video/backlight/88pm860x_bl.c

```
@@ -180,4 +180,5 @@
  data->iset = PM8606_WLED_CURRENT(iset);
  of_property_read_u32(np, "marvell,88pm860x-pwm",
                       &data->pwm);
+ of_node_put(np);
  break;
```

drivers/power/charger-manager.c

```
@@ -1581,8 +1581,10 @@
  cables = devm_kzalloc(dev, sizeof(*cables)
                  * chg_regs->num_cables,
                  GFP_KERNEL);
- if (!cables)
+ if (!cables) {
+       of_node_put(child);
        return ERR_PTR(-ENOMEM);
+ }
```

arch/arm/kernel/devtree.c

```
@@ -101,6 +101,7 @@
  if (of_property_read_u32(cpu, "reg", &hwid)) {
        pr_debug(" * %s missing reg property\n",
                     cpu->full_name);
+       of_node_put(cpu);
        return;
  }

@@ -108,8 +109,10 @@
   * 8 MSBs must be set to 0 in the DT since the reg p
   * defines the MPIDR[23:0].
   */
- if (hwid & ~MPIDR_HWID_BITMASK)
+ if (hwid & ~MPIDR_HWID_BITMASK) {
+       of_node_put(cpu);
        return;
+ }

 /*
  * Duplicate MPIDRs are a recipe for disaster.
```

Challenge: How to make all these changes quickly, consistently and correctly?

Coccinelle to the rescue!

- Allows changes to C code to be expressed using patch-like code patterns (semantic patches) using the language SmPL.
- Applies SmPL semantic patches to an entire code base, updating all relevant code sites at once.
- Under development since 2005. Open source since 2008.

Semantic patches

- Like patches, but independent of irrelevant details
  (line numbers, spacing, variable names, etc.)
- Derived from code, with abstraction.
- Easily adaptable, to eliminate false positives or treat new cases.
- Goal: fit with the existing habits of the Linux programmer.

Semantic patches

- Like patches, but independent of irrelevant details
  (line numbers, spacing, variable names, etc.)
- Derived from code, with abstraction.
- Easily adaptable, to eliminate false positives or treat new cases.
- Goal: fit with the existing habits of the Linux programmer.

No program analysis expertise required.

Example: Find and fix missing `of_node_put` bugs

Memory management in kernel code.

- `kmalloc` to allocate memory.
- `kfree` later, to free it.
- Requires knowing when a free is possible, i.e., no remaining pointers.

Example: Find and fix missing `of_node_put` bugs

Memory management in kernel code.

- `kmalloc` to allocate memory.
- `kfree` later, to free it.
- Requires knowing when a free is possible, i.e., no remaining pointers.

Some parts of the kernel provide more support.

- Reference counting for device nodes.
- `of_node_get` on access.
- `of_node_put` to allow freeing.

Code clutter

`of_node_get`'s and `of_node_put`'s clutter the code.

- Bookkeeping, not functionality.

Solution? Hide them when possible.

```
        for_each_child_of_node(overlay, child)
                adjust_overlay_phandles(child, phandle_delta);
```

Code clutter

`of_node_get`'s and `of_node_put`'s clutter the code.

- Bookkeeping, not functionality.

Solution? Hide them when possible.

```
        for_each_child_of_node(overlay, child)
                adjust_overlay_phandles(child, phandle_delta);
```

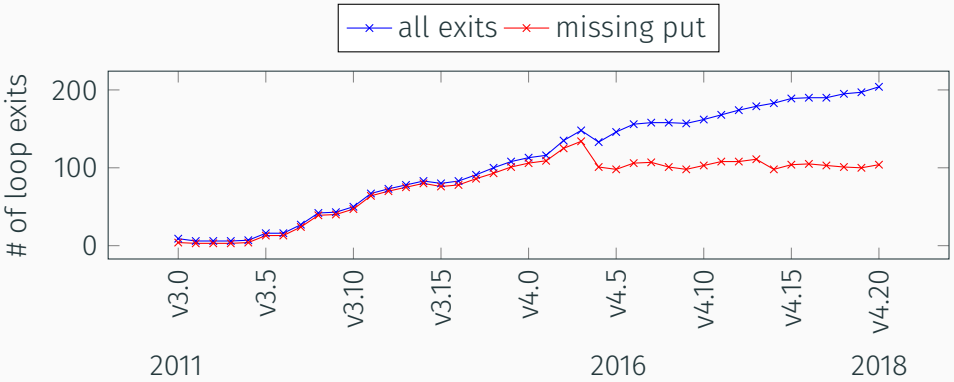
Each iteration increases the reference count of a new element, and decreases the reference count of the previous element.

Problem: What is hidden is easily forgotten

`for_each_child_of_node` hides reference counts in the normal case.

- Problem: Abnormal loop exits don't benefit from the hidden puts:
  - `break`, `return`, `goto`
  - Not `continue`
- Consequence: memory leak.

Evolution of the problem over time



There are currently 9 such iterators

- `for_each_of_cpu_node` added in August 2018.

An example to motivate semantic patch construction

```
  for_each_child_of_node(port, endpoint) {
      if (of_property_read_u32(endpoint, "reg", &endpoint_id))
          endpoint_id = 0;
      if (rockchip_drm_endpoint_is_subdriver(endpoint) > 0)
          continue;
      child_count++;
      ret = drm_of_find_panel_or_bridge(dev->of_node, 0, endpoint_id,
                                        &panel, &bridge);
-     if (!ret)
+     if (!ret) {
+         of_node_put(endpoint);
          break;
+     }
  }
```

Constructing the `of_node_put` semantic patch

Check execution paths through and after the loop body.

```
@@
expression node,child;
iterator name for_each_child_of_node;
@@
for_each_child_of_node(node, child) {
  ...
}
...
```

Constructing the of_node_put semantic patch

All ok if reach an **of_node_put** or add a pointer to **child**.
break requires further attention.

```
@@
expression node,child,e;
iterator name for_each_child_of_node;
@@
for_each_child_of_node(node, child) {
  ...
(
  of_node_put(child);
|
  e = child
|
  break;
)
  ...
}
...
```

Constructing the of_node_put semantic patch

Add `of_node_put` before `break;`.

```
@@
expression node,child,e;
iterator name for_each_child_of_node;
@@
for_each_child_of_node(node, child) {
  ...
(
  of_node_put(child);
|
  e = child
|
+ of_node_put(child);
  break;
)
  ...
}
...
```

Constructing the of_node_put semantic patch

Make **break** optional.

```
@@
expression node,child,e;
iterator name for_each_child_of_node;
@@
for_each_child_of_node(node, child) {
  ...
(
  of_node_put(child);
|
  e = child
|
+ of_node_put(child);
? break;
)
  ...
}
...
```

Constructing the of_node_put semantic patch

Allow any code to follow the matched code in the loop.

By default, the code matched by a pattern adjacent to ... cannot be matched in ...

```
@@
expression node,child,e;
iterator name for_each_child_of_node;
@@
for_each_child_of_node(node, child) {
  ...
(
  of_node_put(child);
|
  e = child
|
+ of_node_put(child);
? break;
)
  ... when any
}
...
```

Constructing the of_node_put semantic patch

Check that **child** is not used after the loop.

```
@@
expression node,child,e;
iterator name for_each_child_of_node;
@@
for_each_child_of_node(node, child) {
  ...
(
  of_node_put(child);
|
  e = child
|
+ of_node_put(child);
? break;
)
  ... when any
}
... when != child
```

Results: Identified missing `of_node_puts` in Linux 4.20 (Dec 2018)

	break	return	goto
`for_each_child_of_node`	4	71	20

Results: Identified missing of_node_puts in Linux 4.20 (Dec 2018)

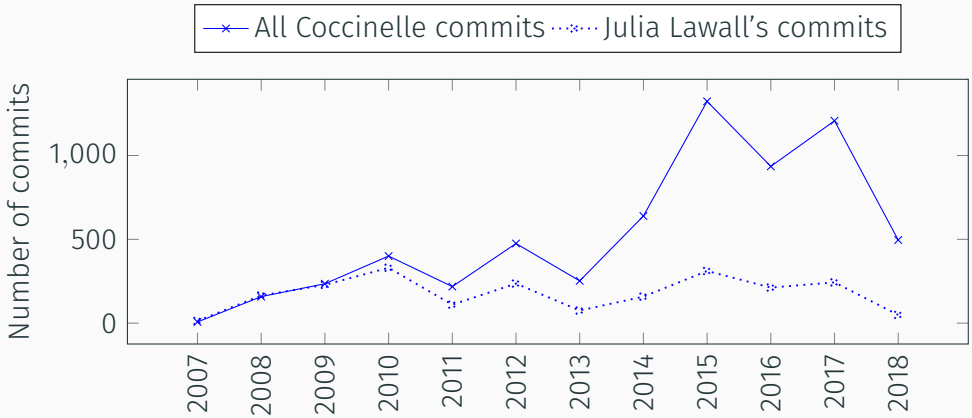
	break	return	goto
for_each_child_of_node	4	71	20
for_each_node_by_name	1	1	0
for_each_node_by_type	1	1	0
for_each_compatible_node	2	7	0
for_each_matching_node	2	3	1
for_each_matching_node_and_match	1	8	1
for_each_available_child_of_node	19	23	18
for_each_node_with_property	0	0	0
for_each_of_cpu_node	0	0	0

Results: Identified missing of_node_puts in Linux 4.20 (Dec 2018)

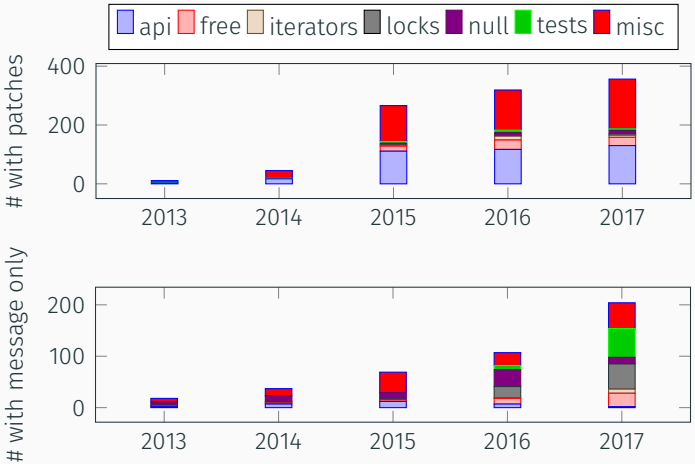
	break	return	goto
for_each_child_of_node	4	71	20
for_each_node_by_name	1	1	0
for_each_node_by_type	1	1	0
for_each_compatible_node	2	7	0
for_each_matching_node	2	3	1
for_each_matching_node_and_match	1	8	1
for_each_available_child_of_node	19	23	18
for_each_node_with_property	0	0	0
for_each_of_cpu_node	0	0	0

Studying these results reveals other types of reference count problems for which we can write semantic patches.

Impact: Coccinelle-related Linux kernel commits per year



Impact: Intel's 0-day test service



Scaling up to Linux development

- How to write semantic patches?
- How to validate the results?
- How to submit the resulting patches?
- Scaling to multiple development streams.

How to write semantic patches?

Kernel developers are C programmers, not SmPL programmers.

- Some use Coccinelle regularly, and become experts
  - Thomas Gleixner, Kees Cook, etc.
- Others make widespread changes infrequently.
  - Learn/relearn SmPL on their own.
  - Ask us to solve the problem (`of_node_put`).
  - Ask another developer to solve the problem.
  - Make the change without Coccinelle, and make a note to do better next time.

How to write semantic patches?

Kernel developers are C programmers, not SmPL programmers.

- Some use Coccinelle regularly, and become experts
  - Thomas Gleixner, Kees Cook, etc.
- Others make widespread changes infrequently.
  - Learn/relearn SmPL on their own.
  - Ask us to solve the problem (`of_node_put`).
  - Ask another developer to solve the problem.
  - Make the change without Coccinelle, and make a note to do better next time.

Spinfer: Infer a semantic patch from a few change examples (in progress).

How to validate changes?

Coccinelle makes it easy to make lots of changes, very fast.

- Changes needed validation, whether or not generated using Coccinelle.
- Testing is hard, may need unavailable hardware, specific inputs.
- Even compilation is hard, due to configuration options.

How to validate changes?

make coccicheck

- Make target in the Linux kernel, currently running around 60 bug-finding/fixing semantic patches.

How to validate changes?

make coccicheck

- Make target in the Linux kernel, currently running around 60 bug-finding/fixing semantic patches.

0-Day, from Intel: compilation for many configurations and build testing.

- Runs a selection of Coccinelle scripts, via make coccicheck.
- Almost 600 0-day reports motivated by Coccinelle rules in 2017.

How to validate changes?

make coccicheck

- Make target in the Linux kernel, currently running around 60 bug-finding/fixing semantic patches.

0-Day, from Intel: compilation for many configurations and build testing.

- Runs a selection of Coccinelle scripts, via make coccicheck.
- Almost 600 0-day reports motivated by Coccinelle rules in 2017.

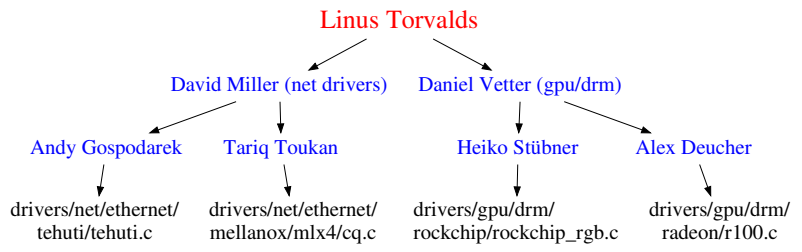
JMake: Make for kernel janitors [DSN 2017].

- Chooses a relevant configuration and reports whether changed code was subjected to the compiler.

How to submit changes?

Coccinelle makes it easy to make lots of changes, very fast.

- Changes need to go to the right person in the maintainer hierarchy.



How to submit changes?

get_maintainer.pl: Given a patch, returns the list of relevant maintainers.

- Introduced by Joe Perches in 2009.

How to submit changes?

get_maintainer.pl: Given a patch, returns the list of relevant maintainers.

- Introduced by Joe Perches in 2009.

splitpatch: Tool bundled with Coccinelle to split a diff by maintainer and then construct appropriate email messages.

How to submit changes?

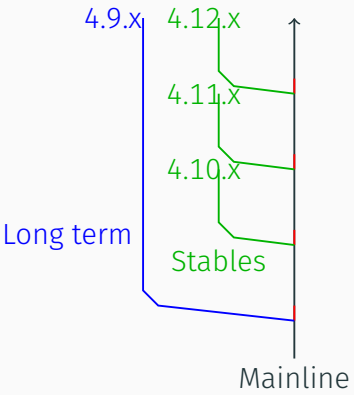
get_maintainer.pl: Given a patch, returns the list of relevant maintainers.

- Introduced by Joe Perches in 2009.

splitpatch: Tool bundled with Coccinelle to split a diff by maintainer and then construct appropriate email messages.

Coccinelle has been suggested as a solution to synchronize treewide changes with the current kernel version.

Challenge: Scaling to multiple development streams



- Backports project uses Coccinelle to retarget version specific code to a version generic library [EDCC 2015]
- Prequel finds commits that illustrate how to port drivers across versions [USENIX ATC 2017]
- Spinfer infers semantic patches from such examples

Conclusion

- Coccinelle, provides user scriptable matching and transformation of C code.
- Lessons learned:
  - Take the expertise of the target users into account.
  - Avoid creeping featurism: Do one thing and do it well.
- Success measures:
  - Over 6600 commits in the Linux kernel based on Coccinelle.
  - EuroSys test of time award (2018).

Conclusion

- Coccinelle, provides user scriptable matching and transformation of C code.
- Lessons learned:
  - Take the expertise of the target users into account.
  - Avoid creeping featurism: Do one thing and do it well.
- Success measures:
  - Over 6600 commits in the Linux kernel based on Coccinelle.
  - EuroSys test of time award (2018).
- Probably, everyone in this room uses some Coccinelle modified code!

Conclusion

- Coccinelle, provides user scriptable matching and transformation of C code.
- Lessons learned:
  - Take the expertise of the target users into account.
  - Avoid creeping featurism: Do one thing and do it well.
- Success measures:
  - Over 6600 commits in the Linux kernel based on Coccinelle.
  - EuroSys test of time award (2018).
- Probably, everyone in this room uses some Coccinelle modified code!

http://coccinelle.lip6.fr