The List is Our Process!
An analysis of the kernel's email-based development process

Ralf Ramsauer*, Sebastian Duda†, Lukas Bulwahn‡, Wolfgang Mauerer*§

* Technical University of Applied Sciences Regensburg
† Friedrich–Alexander University Erlangen–Nürnberg
‡ Hobbyist, active in LF ELISA Project, employed at BMW AG
§ Siemens AG, Corporate Research and Technology, Munich

Linux Plumbers Conference, Lisbon

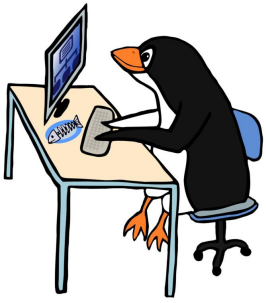
September 9, 2019

OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

SIEMENS

Our Overall Goal

Formalising and assessing the Linux Kernel development process



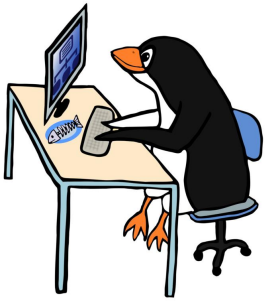
© Moini
openclipart.org

Our Overall Goal

Formalising and assessing the Linux Kernel development process

Outside / Inside Motivation

- Safety-Critical Development
- Development Process Assessment
- Monitoring (cf. CHAOSS)
- Fundamentals of Software Engineering



Motivation from *outside* the community

Academic Research

- Applying statistical (big data, ML, etc.) methods on software development projects
- Linux kernel serves as a prime example for their evaluation

Commercial Users

- Measuring progress of certain long-term kernel developments, e.g., mainlining the PREEMPT_RT patch
- Measuring and monitoring many business-critical open-source projects a scale (CHAOSS)
- Fulfilling the certification requirement "Development process assessment" in regulated environments (safety- and security-related certified systems)

Motivation from *inside* the community

Interest of the kernel community itself

- D. Williams, Towards a Linux Kernel Maintainer Handbook, LPC 2018
- J. Corbet, Change IDs for kernel patches, `https://lwn.net/Articles/797613/`
- [Ksummit-discuss] [MAINTAINERS SUMMIT] Patch version changes in commit logs?
- [Ksummit-discuss] Allowing something Change-Id (or something like it) in kernel commits

Towards a Linux Kernel Maintainer Handbook

Dan Williams

Linux Plumbers 2018

Subsystem Profile

- Describe maintenance of your sub-system as a “bus-factor” document
- Document the policies that each subsystem typical decides differently

Patches or Pull requests
Last day for new feature submissions
Last day to merge features
Non-author Ack / Review Tags Required
Test Suite

Trusted Reviewers
Resubmit Cadence
Time Zone / Office Hours
Checkpatch / Style cleanups
Off-list review

Others?

Subsystem Profile

- Describe maintenance of your sub-system as a “bus-factor” document
- Document the policies that each subsystem typical decides differently

Patches or Pull requests
Last day for new feature submissions
Last day to merge features
Non-author Ack / Review Tags Required
Test Suite

Trusted Reviewers
Resubmit Cadence
Time Zone / Office Hours
Checkpatch / Style cleanups
Off-list review

Others?

Subsystem Profile

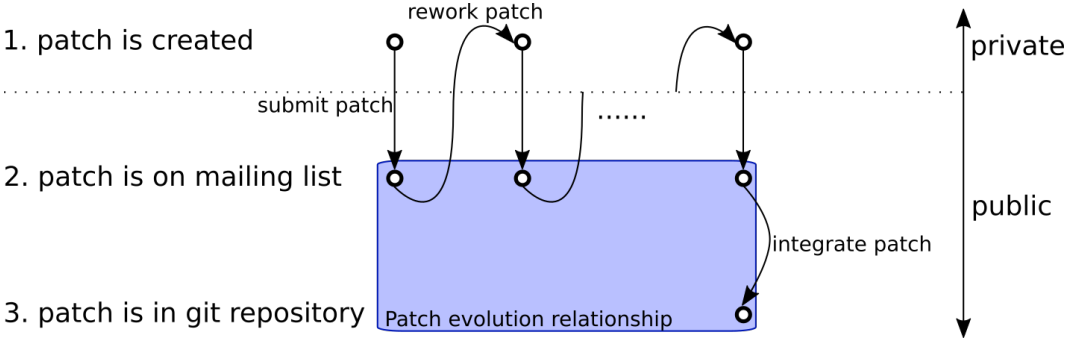
- Describe maintenance of your sub-system as a “bus-factor” document
- Document the policies that each subsystem typical decides differently

Patches or Pull requests
Last day for new feature submissions
Last day to merge features
Non-author Ack / Review Tags Required
Test Suite

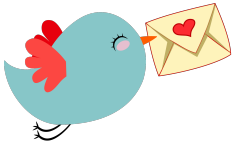
Trusted Reviewers
Resubmit Cadence
Time Zone / Office Hours
Checkpatch / Style cleanups
Off-list review

Others?

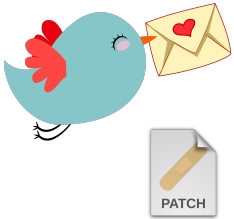
Towards a formal model of the development process



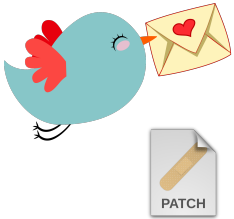
Linux Kernel development workflow



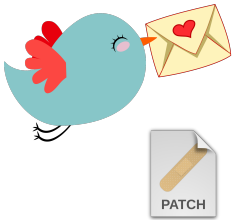
Linux Kernel development workflow

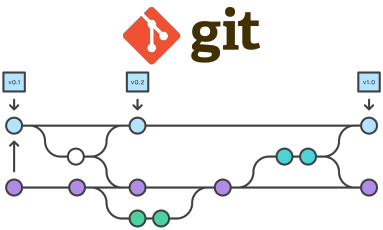


Linux Kernel development workflow

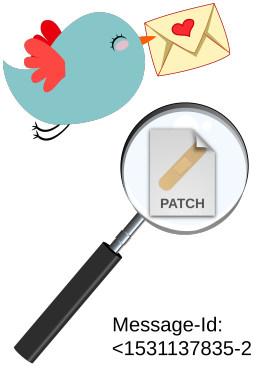


Linux Kernel development workflow

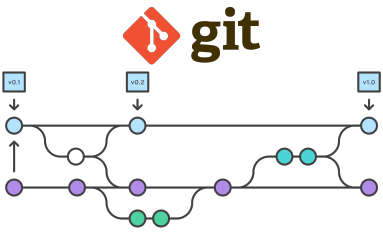




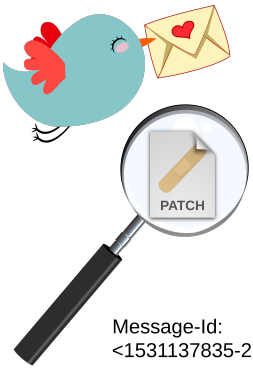
Linux Kernel development workflow



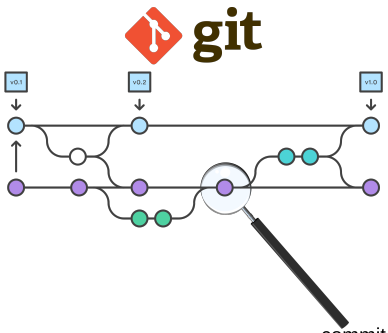
Message-Id:
<1531137835-21581-1-git@1wt.eu>



Linux Kernel development workflow

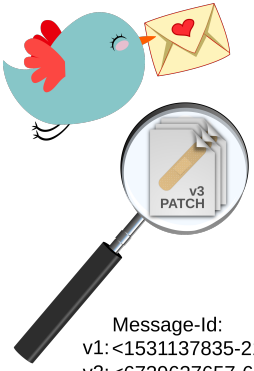


Message-Id:
<1531137835-21581-1-git@1wt.eu>

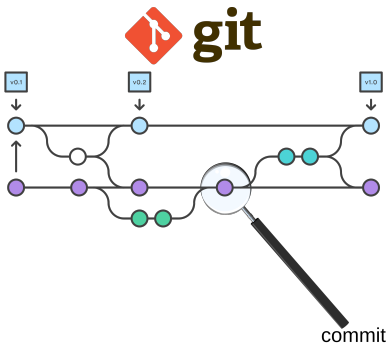


commit
2f6e24d3151fb9967774f9721b288f216f3180df

Linux Kernel development workflow

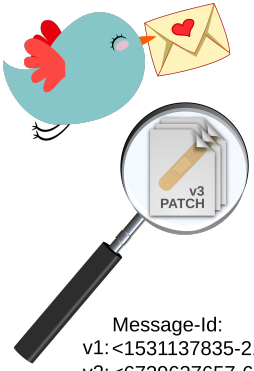


Message-Id:
v1: <1531137835-21581-1-git@1wt.eu>
v2: <6739637657-68462-1-git@1wt.eu>
v3: <9717683099-75474-1-git@1wt.eu>

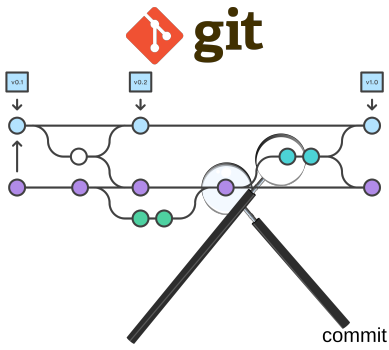


commit
2f6e24d3151fb9967774f9721b288f216f3180df

Linux Kernel development workflow

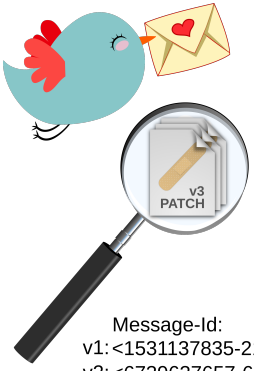


Message-Id:
v1: <1531137835-21581-1-git@1wt.eu>
v2: <6739637657-68462-1-git@1wt.eu>
v3: <9717683099-75474-1-git@1wt.eu>

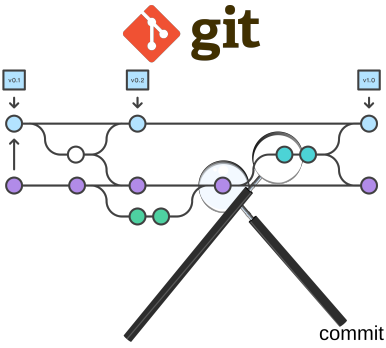


commit
2f6e24d3151fb9967774f9721b288f216f3180df
89876f275e8d562912d9c238cd888b52065cf25

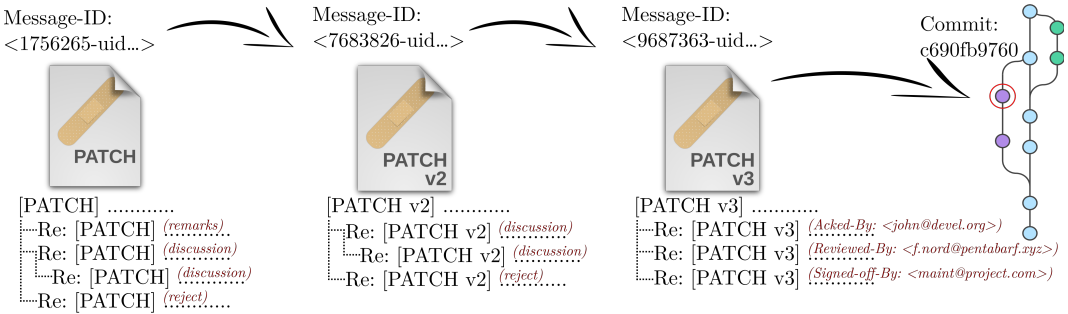
Linux Kernel development workflow



Message-Id:
v1: <1531137835-21581-1-git@1wt.eu>
v2: <6739637657-68462-1-git@1wt.eu>
v3: <9717683099-75474-1-git@1wt.eu>



commit
2f6e24d3151fb9967774f9721b288f216f3180df
89876f275e8d562912d9c238cd888b52065cf25


Message-ID:
<1756265-uid...>
PATCH
[PATCH] .....
Re: [PATCH] (remarks).....
Re: [PATCH] (discussion).....
Re: [PATCH] (discussion).....
Re: [PATCH] (reject).....
Message-ID:
<7683826-uid...>
PATCH v2
[PATCH v2] .....
Re: [PATCH v2] (discussion).....
Re: [PATCH v2] (discussion).....
Re: [PATCH v2] (reject).....
Message-ID:
<9687363-uid...>
PATCH v3
[PATCH v3] .....
Re: [PATCH v3] (Acked-By: <john@devel.org>)
Re: [PATCH v3] (Reviewed-By: <f.nord@pentabarf.xyz>)
Re: [PATCH v3] (Signed-off-By: <maint@project.com>)
Commit:
c690fb9760

PaStA - Patch Stack Analysis

- Detects *similar* patches across different branches
- Quantify mainlining efforts of off-tree developments (Preempt_RT, vendor kernels, . . .)
- Works with mailing lists!

Source: toplock.net.au

Example of similar patches

```
commit 91824d74d6d85f58c63a66b8f2c7993ae246181b
Author: Thomas Gleixner <tglx@linutronix.de>
Date:   Mon Sep 12 21:45:49 2011 +0200

sched-cure-utter-idle-accounting-madness.patrch

Signed-off-by: Thomas Gleixner <tglx@linutronix.de>

diff --git a/kernel/sched.c b/kernel/sched.c
index 205499a..1121a97 100644
--- a/kernel/sched.c
+++ b/kernel/sched.c
@@ -5037,7 +5037,13 @@ EXPORT_SYMBOL(task_nice);
  */
 int idle_cpu(int cpu)
 {
-       return cpu_curr(cpu) == cpu_rq(cpu)->idle;
+       struct rq *rq = cpu_rq(cpu);
+
+#ifdef CONFIG_SMP
+       return rq->curr == rq->idle && !rq->nr_running && !rq->wake_list;
+#else
+       return rq->curr == rq->idle && !rq->nr_running;
+#endif
 }

 /**
```

```
commit 908a3283728d92df36e0c7cd63304fd35e93a8a9
Author: Thomas Gleixner <tglx@linutronix.de>
Date:   Thu Sep 15 15:32:06 2011 +0200

sched: Fix idle_cpu()

On -rt we observed hackbench waking all 400 tasks to a single
cpu. This is because of select_idle_sibling()'s interaction
with the new ipi based wakeup scheme.
[..snip..]
Signed-off-by: Thomas Gleixner <tglx@linutronix.de>
Signed-off-by: Peter Zijlstra <a.p.zijlstra@chello.nl>
Link: http://lkml.kernel.org/n/tip-3b30p18b2[...]
Signed-off-by: Ingo Molnar <mingo@elte.hu>

diff --git a/kernel/sched.c b/kernel/sched.c
index 1874c74..4cdc91c 100644
--- a/kernel/sched.c
+++ b/kernel/sched.c
@@ -5138,7 +5138,20 @@ EXPORT_SYMBOL(task_nice);
  */
 int idle_cpu(int cpu)
 {
-       return cpu_curr(cpu) == cpu_rq(cpu)->idle;
+       struct rq *rq = cpu_rq(cpu);
+
+       if (rq->curr != rq->idle)
+               return 0;
+
+       if (rq->nr_running)
+               return 0;
+
+#ifdef CONFIG_SMP
+       if (!llist_empty(&rq->wake_list))
+               return 0;
+#endif
+
+       return 1;
 }

 /**
```

Example of similar patches

commit 91824d74d6d85f58c63a66b8f2c7993ae246181b
Author: Thomas Gleixner <tglx@linutronix.de>
Date: Mon Sep 12 21:45:49 2011 +0200

sched-cure-utter-idle-accounting-madness.patrch

Signed-off-by: Thomas Gleixner <tglx@linutronix.de>

diff --git a/kernel/sched.c b/kernel/sched.c
index 205499a..1121a97 100644

```
+       struct rq *rq = cpu_rq(cpu);
+
+#ifdef CONFIG_SMP
+       return rq->curr == rq->idle && !rq->nr_r
+#else
+       return rq->curr == rq->idle && !rq->nr_r
+#endif
```

}

/**

commit 908a3283728d92df36e0c7cd63304fd35e93a8a9
Author: Thomas Gleixner <tglx@linutronix.de>
Date: Thu Sep 15 15:32:06 2011 +0200

sched: Fix idle_cpu()

On -rt we observed hackbench waking all 400 tasks to a single cpu. This is because of select_idle_sibling()'s interaction with the new ipi based wakeup scheme.
[..snip..]
Signed-off-by: Thomas Gleixner <tglx@linutronix.de>

```
+       struct rq *rq = cpu_rq(cpu);
+
+       if (rq->curr != rq->idle)
+               return 0;
+
+       if (rq->nr_running)
+               return 0;
+
+#ifdef CONFIG_SMP
+       if (!llist_empty(&rq->wake_list))
+               return 0;
+#endif
+
+       return 1;
```

Example of similar patches

```
commit 91824d74d6d85f58c63a66b8f2c7993ae246181b
Author: Th                                    >
Date:   Mo

sched–cure-                                   atrch

Signed–off-                                   onix.de>

 diff ––git                                   c
 index 2054
––– a/kern
+++ b/kern
@@ –5037,7                                    nice);
   */
  int idle_
  {
–         re                                  –>idle;
+         st
+
+#ifdef CO
+         re                                  >nr_running && !rq–>wake_list;
+#else
+         return rq–>curr == rq–>idle && !rq–>nr_running;
+#endif
  }

  /**
```

```
[...]
#else
#endif
#ifdef
CONFIG_SMP
cpu_rq(cpu)
return
rq
rq–>curr
rq–>idle
rq–>nr_running
rq–>wake_list
struct
```

```
commit 908a3283728d92df36e0c7cd63304fd35e93a8a9
Author: Thomas Gleixner <tglx@linutronix.de>
Date:   Thu Sep 15 15:32:06 2011 +0200

sched: Fix idle_cpu()

On –rt we observed hackbench waking all 400 tasks to a single
cpu. Th                                       ing()'s interaction
with the
[..snip
Signed–                                       utronix.de>
Signed–                                       ra@chello.nl>
Link: h                                       18b2[...]
Signed–                                       u>

 diff ––                                      ed.c
 index 18
––– a/k
+++ b/k
@@ –513                                       k_nice);
   */
  int idl
  {
–                                             pu)–>idle;
+
+
+
+
+
+                 return 0;
+
+#ifdef CONFIG_SMP
+         if (!llist_empty(&rq–>wake_list))
+                 return 0;
+#endif
+
+         return 1;
  }

  /**
```

```
[...]
#endif
#ifdef
if
CONFIG_SMP
cpu_rq(cpu)
llist_empty
return
rq
rq–>curr
rq–>idle
rq–>nr_running
rq–>wake_list
struct
```

Example of similar patches

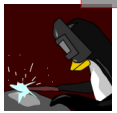
```
commit 91824d74d6d85f58c63a66b8f2c7993ae246181b
Author: Th                                    >
Date:   Mo

sched–cure-                                   atrch

Signed–off-                                   onix.de>

 diff ––git                                   c
 index 2054
––– a/kern
+++ b/kern
@@ –5037,7                                    nice);
   */
  int idle_
  {
–         re                                  –>idle;
+         st
+
+#ifdef CO
+         re                                  >nr_running && !rq–>wake_list;
+#else
+         return rq–>curr == rq–>idle && !rq–>nr_running;
+#endif
  }

  /**
```

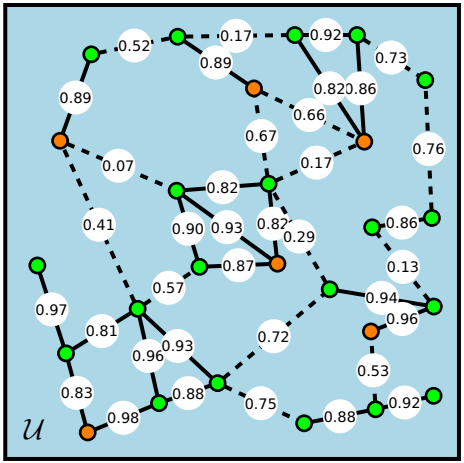
```
[...]
#else
#endif
#ifdef
CONFIG_SMP
cpu_rq(cpu)
return
rq
rq–>curr
rq–>idle
rq–>nr_running
rq–>wake_list
struct
```

```
commit 908a3283728d92df36e0c7cd63304fd35e93a8a9
Author: Thomas Gleixner <tglx@linutronix.de>
Date:   Thu Sep 15 15:32:06 2011 +0200

sched: Fix idle_cpu()

On –rt we observed hackbench waking all 400 tasks to a single
cpu. Th                                       ing()'s interaction
with the
[..snip
Signed–                                       utronix.de>
Signed–                                       ra@chello.nl>
Link: h                                       18b2[...]
Signed–                                       u>

 diff ––                                      ed.c
 index 18
––– a/k
+++ b/k
@@ –513                                       k_nice);
   */
  int idl
  {
–                                             pu)–>idle;
+
+
+
+
+
+                 return 0;
+
+                 return 0;
+#endif
+
+         return 1;
  }

  /**
```

```
[...]
#endif
#ifdef
if
CONFIG_SMP
cpu_rq(cpu)
llist_empty
return
rq
rq–>curr
rq–>idle
rq–>nr_running
rq–>wake_list
struct
```

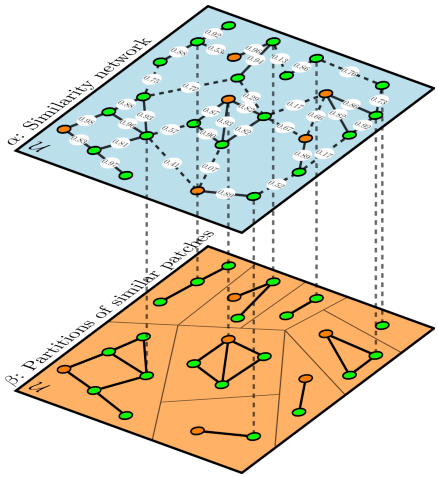
Diff similarity: 0.875





Legend

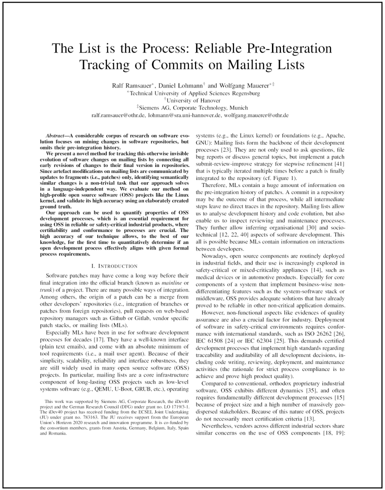
- green nodes: patches on MLs
- orange nodes: commits in repository
- edges: similarity of patches/commits
  - dashed: similarity below thres
  - solid: similarity above thres



Legend

- green nodes: patches on MLs
- orange nodes: commits in repository
- edges: similarity of patches/commits
  - dashed: similarity below thres
  - solid: similarity above thres

Interested in the techniques? More details in:



The List is the Process: Reliable Pre-Integration Tracking of Commits on Mailing Lists

Ralf Ramsauer*, Daniel Lohmann† and Wolfgang Mauerer*‡
*Technical University of Applied Sciences Regensburg
†University of Hanover
‡Siemens AG, Corporate Technology, Munich
ralf.ramsauer@othr.de, lohmann@sra.uni-hannover.de, wolfgang.mauerer@othr.de

Abstract—**A considerable corpus of research on software evolution focuses on mining changes in software repositories, but omits their pre-integration history.**

We present a novel method for tracking this otherwise invisible evolution of software changes on mailing lists by connecting all early revisions of changes to their final version in repositories. Since artefact modifications on mailing lists are communicated by updates to fragments (i.e., patches) only, identifying semantically similar changes is a non-trivial task that our approach solves in a language-independent way. We evaluate our method on high-profile open source software (OSS) projects like the Linux kernel, and validate its high accuracy using an elaborately created ground truth.

Our approach can be used to quantify properties of OSS development processes, which is an essential requirement for using OSS in reliable or safety-critical industrial products, where certifiability and conformance to processes are crucial. The high accuracy of our technique allows, to the best of our knowledge, for the first time to quantitatively determine if an open development process effectively aligns with given formal process requirements.

I. Introduction

Software patches may have come a long way before their final integration into the official branch (known as *mainline* or *trunk*) of a project. There are many possible ways of integration. Among others, the origin of a patch can be a merge from other developers' repositories (i.e., integration of branches or patches from foreign repositories), pull requests on web-based repository managers such as Github or Gitlab, vendor specific patch stacks, or mailing lists (MLs).

Especially MLs have been in use for software development processes for decades [17]. They have a well-known interface (plain text emails), and come with an absolute minimum of tool requirements (i.e., a mail user agent). Because of their simplicity, scalability, reliability and interface robustness, they are still widely used in many open source software (OSS) projects. In particular, mailing lists are a core infrastructure component of long-lasting OSS projects such as low-level systems software (e.g., QEMU, U-Boot, GRUB, etc.), operating systems (e.g., the Linux kernel) or foundations (e.g., Apache, GNU): Mailing lists form the backbone of their development processes [23]. They are not only used to ask questions, file bug reports or discuss general topics, but implement a patch submit-review-improve strategy for stepwise refinement [41] that is typically iterated multiple times before a patch is finally integrated to the repository (cf. Figure 1).

Therefore, MLs contain a huge amount of information on the pre-integration history of patches. A commit in a repository may be the outcome of that process, while all intermediate steps leave no direct traces in the repository. Mailing lists allow us to analyse development history and code evolution, but also enable us to inspect reviewing and maintenance processes. They further allow inferring organisational [30] and socio-technical [12, 22, 40] aspects of software development. This all is possible because MLs contain information on interactions between developers.

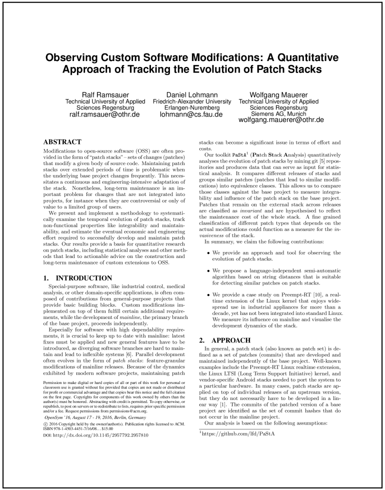
Nowadays, open source components are routinely deployed in industrial fields, and their use is increasingly explored in safety-critical or mixed-criticality appliances [14], such as medical devices or in automotive products. Especially for core components of a system that implement business-wise non-differentiating features such as the system-software stack or middleware, OSS provides adequate solutions that have already proved to be reliable in other non-critical application domains.

However, non-functional aspects like evidences of quality assurance are also a crucial factor for industry. Deployment of software in safety-critical environments requires conformance with international standards, such as ISO 26262 [26], IEC 61508 [24] or IEC 62304 [25]. This demands certified development processes that implement high standards regarding traceability and auditability of all development decisions, including code writing, reviewing, deployment, and maintenance activities (the rationale for strict process compliance is to achieve and prove high product quality).

Compared to conventional, orthodox proprietary industrial software, OSS exhibits different dynamics [35], and often requires fundamentally different development processes [15] because of project size and a high number of massively geo-dispersed stakeholders. Because of this nature of OSS, projects do not necessarily meet certification criteria [13].

Nevertheless, vendors across different industrial sectors share similar concerns on the use of OSS components [18, 19]:

This work was supported by Siemens AG, Corporate Research, the iDev40 project and the German Research Council (DFG) under grant no. LO 1719/3-1. The iDev40 project has received funding from the ECSEL Joint Undertaking (JU) under grant no. 783163. The JU receives support from the European Union's Horizon 2020 research and innovation programme. It is co-funded by the consortium members, grants from Austria, Germany, Belgium, Italy, Spain and Romania.



Observing Custom Software Modifications: A Quantitative Approach of Tracking the Evolution of Patch Stacks

Ralf Ramsauer, Technical University of Applied Sciences Regensburg, ralf.ramsauer@othr.de
Daniel Lohmann, Friedrich-Alexander University Erlangen-Nuremberg, lohmann@cs.fau.de
Wolfgang Mauerer, Technical University of Applied Sciences Regensburg, Siemens AG, Munich, wolfgang.mauerer@othr.de

ABSTRACT

Modifications to open-source software (OSS) are often provided in the form of "patch stacks" – sets of changes (patches) that modify a given body of source code. Maintaining patch stacks over extended periods of time is problematic when the underlying base project changes frequently. This necessitates a continuous and engineering-intensive adaptation of the stack. Nonetheless, long-term maintenance is an important problem for changes that are not integrated into projects, for instance when they are controversial or only of value to a limited group of users.

We present and implement a methodology to systematically examine the temporal evolution of patch stacks, track non-functional properties like integrability and maintainability, and estimate the eventual economic and engineering effort required to successfully develop and maintain patch stacks. Our results provide a basis for quantitative research on patch stacks, including statistical analyses and other methods that lead to actionable advice on the construction and long-term maintenance of custom extensions to OSS.

1. INTRODUCTION

Special-purpose software, like industrial control, medical analysis, or other domain-specific applications, is often composed of contributions from general-purpose projects that provide basic building blocks. Custom modifications implemented on top of them fulfill certain additional requirements, while the development of *mainline*, the primary branch of the base project, proceeds independently.

Especially for software with high dependability requirements, it is crucial to keep up to date with mainline: latest fixes must be applied and new general features have to be introduced, as diverging software branches are hard to maintain and lead to inflexible systems [6]. Parallel development often evolves in the form of *patch stacks*: feature-granular modifications of mainline releases. Because of the dynamics exhibited by modern software projects, maintaining patch stacks can become a significant issue in terms of effort and costs.

Our toolkit PaStA[1] (**Pa**tch **St**ack **A**nalysis) quantitatively analyses the evolution of patch stacks by mining git [5] repositories and produces data that can serve as input for statistical analysis. It compares different releases of stacks and groups similar patches (patches that lead to similar modifications) into equivalence classes. This allows us to compare those classes against the base project to measure integrability and influence of the patch stack on the base project. Patches that remain on the external stack across releases are classified as *invariant* and are hypothesised to reflect the maintenance cost of the whole stack. A fine grained classification of different patch types that depends on the actual modifications could function as a measure for the *invasiveness* of the stack.

In summary, we claim the following contributions:

- We provide an approach and tool for observing the evolution of patch stacks.
- We propose a language-independent semi-automatic algorithm based on string distances that is suitable for detecting similar patches on patch stacks.
- We provide a case study on Preempt-RT [10], a real-time extension of the Linux kernel that enjoys wide-spread use in industrial appliances for more than a decade, yet has not been integrated into standard Linux. We measure its influence on mainline and visualise the development dynamics of the stack.

2. APPROACH

In general, a patch stack (also known as patch set) is defined as a set of patches (commits) that are developed and maintained independently of the base project. Well-known examples include the Preempt-RT Linux realtime extension, the Linux LTSI (Long Term Support Initiative) kernel, and vendor-specific Android stacks needed to port the system to a particular hardware. In many cases, patch stacks are applied on top of individual releases of an upstream version, but they do not necessarily have to be developed in a linear way [1]. The commits of the patched version of a base project are identified as the set of commit hashes that do not occur in the mainline project.

Our analysis is based on the following assumptions:

[1] https://github.com/lfd/PaStA

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
OpenSym '16, August 17 - 19, 2016, Berlin, Germany
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4451-7/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2957792.2957810

Data Acquisition

- Dumps from gmane.org etc.
- kernel.org public inboxes
  - some lists, prehistoric data
  - `https://lore.kernel.org/lists.html`
- Our own collection
  - 200 lists, since May ’19
  - `https://github.com/linux-mailinglist-archives`

Let there be chaos

- Br?ken encoding
- BÃse64
-
- MUAs
- Bots
- HTML
- Automated mails
- non-Linux patches
- Stable reviews
- Malformed recipients
- ...

Message-Id: <74851t0$h3103wn0$Delldi Fri, 9 Mar 71685 18:45:56 +0000

Date: Mon, 08 Aug 05 04:01:15 ?x?_???????

Date: Tue, 27 Mar 22001 13:42:39 +0200 (Westeurop?ische Sommerzeit)

X-Mailer: Microsoft Outlook Express 6.00.2900.3028

We analyse...

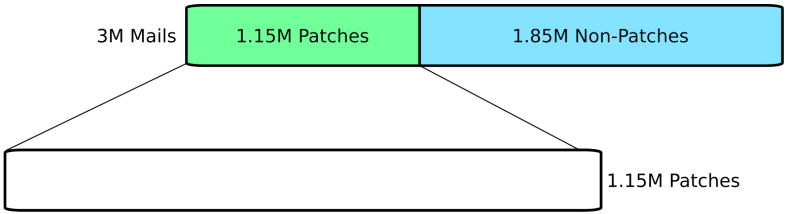
- v2.6.39..linus/master
- ≈610K commits
- Mails: 2011-05-01 – 2018-12-31
- ≈3M mails
- Lists: All Public Inboxes from lore.kernel.org

We analyse...

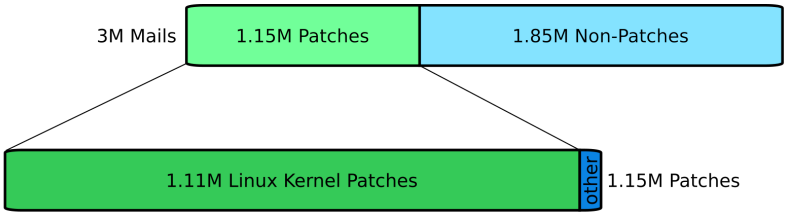
- v2.6.39..linus/master
- ≈610K commits
- Mails: 2011-05-01 – 2018-12-31
- ≈3M mails
- Lists: All Public Inboxes from lore.kernel.org
- linux-amlogic, **linux-arm-kernel**, linux-i3c, linux-mtd, linux-riscv, linuxppc-dev, cocci, linux-block, linux-bluetooth, linux-btrfs, linux-cifs, linux-clk, linux-crypto, linux-ext4, linux-fsdevel, linux-hwmon, linux-iio, linux-integrity, **linux-kernel**, linux-media, linux-mips, linux-modules, **linux-next**, **netdev**, linux-nfs, linux-parisc, linux-pci, linux-renesas-soc, linux-rtc, linux-security-module, linux-sgx, linux-trace-devel, linux-watchdog, linux-wireless

2011-05–2018-12

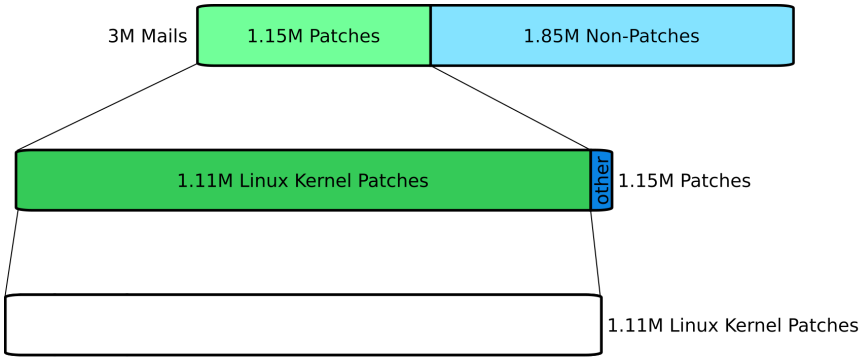
2011-05–2018-12



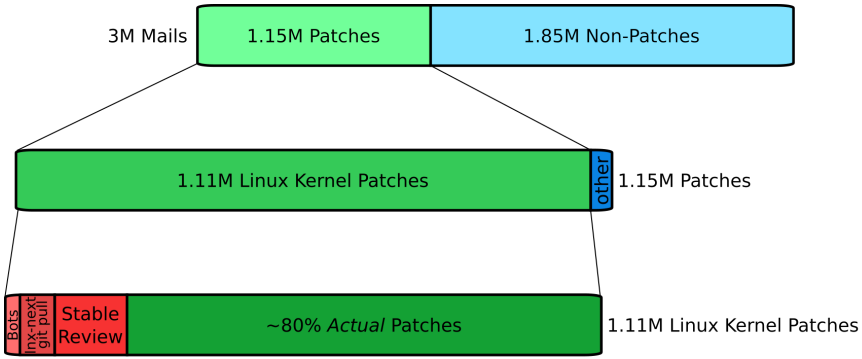
2011-05–2018-12



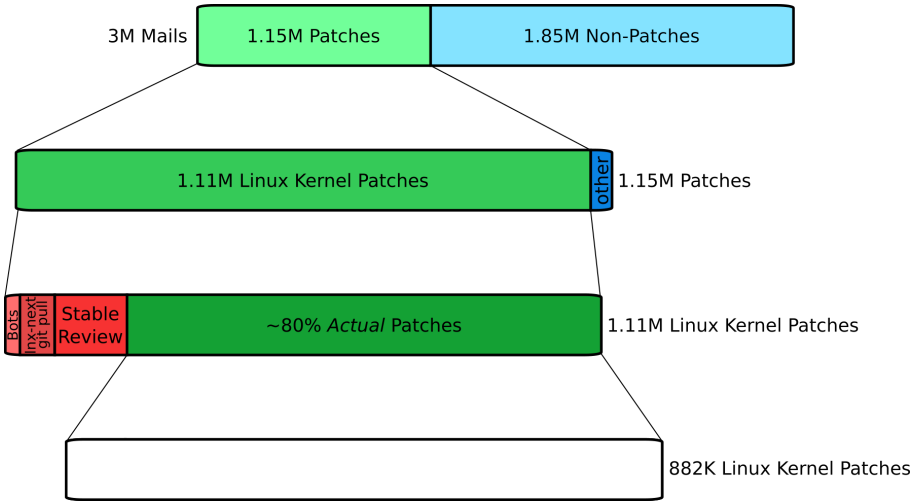
2011-05–2018-12



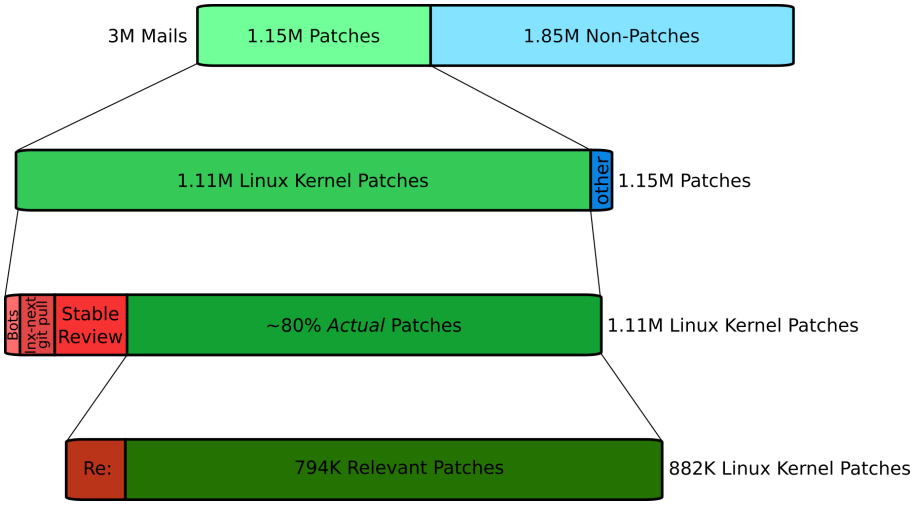
2011-05–2018-12



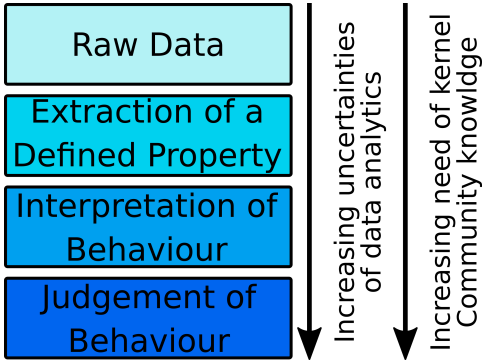
2011-05–2018-12



2011-05–2018-12



Increasing Levels of Data Aggregation



Our main work

- Focus on the first two levels and *selectively* interpret observations to *describe* behaviour
- We do **NOT** try and **NOT** intend to judge behaviour

Ignored Patches

Definition

A patch on a ML is *ignored* if...

- ... the thread of the patch has no responses from persons other than the author
- ... the patch was not accepted upstream
- ... all related patches (e.g., revisions in other series) were ignored

Research Question

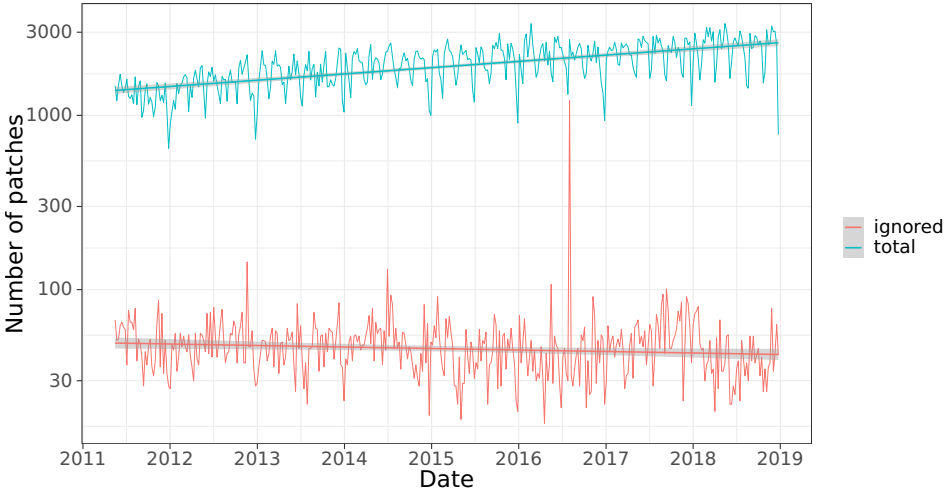
Are there specific characteristics for ignored patches?

Ignored Patches

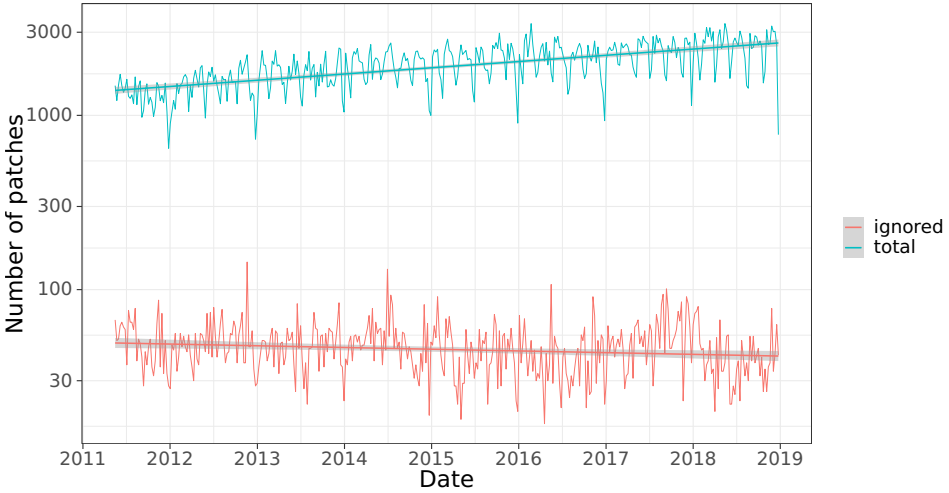
By the numbers...

- lore.kernel.org lists 2011-2018: ø2.5%
  - 2011: ø3.9%
  - 2015: ø2.1%
  - 2018: ø1.6%
- vs. ALL mailing lists: (data 2019-05 – today)
  - ø3.3%

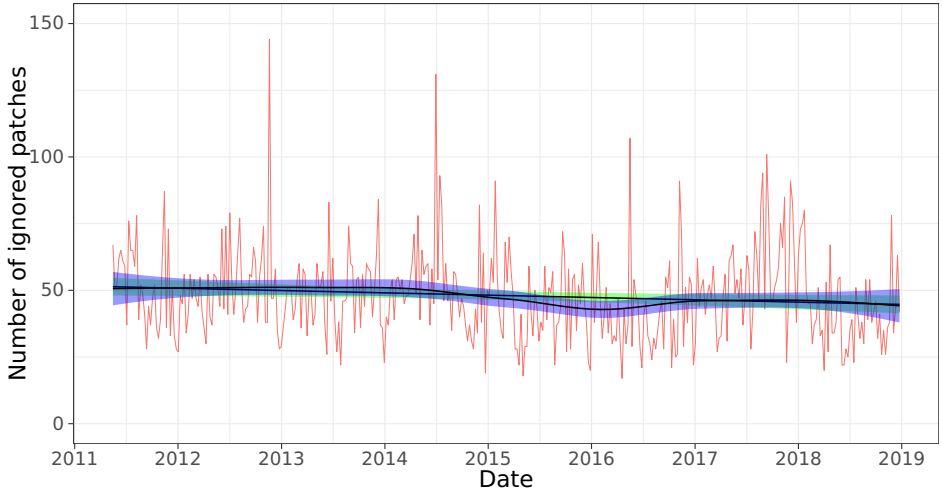
Evolution of ignored patches



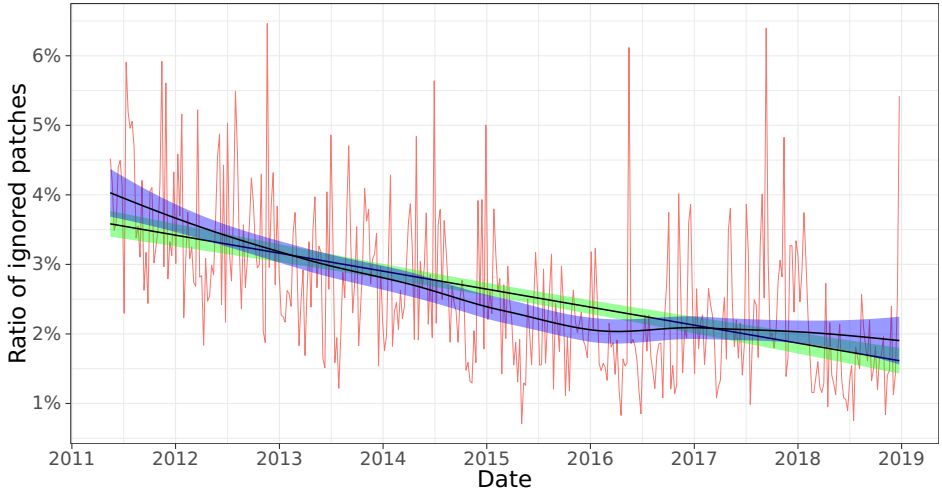
Evolution of ignored patches



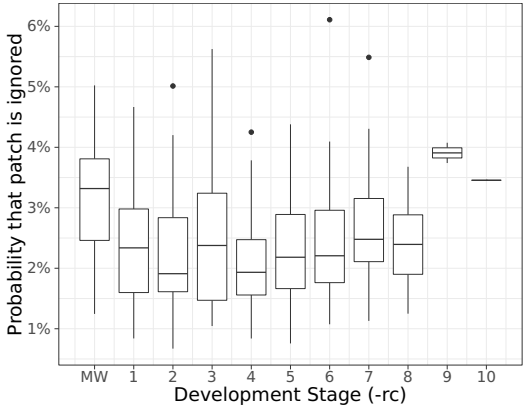
Evolution of ignored patches



Evolution of ignored patches



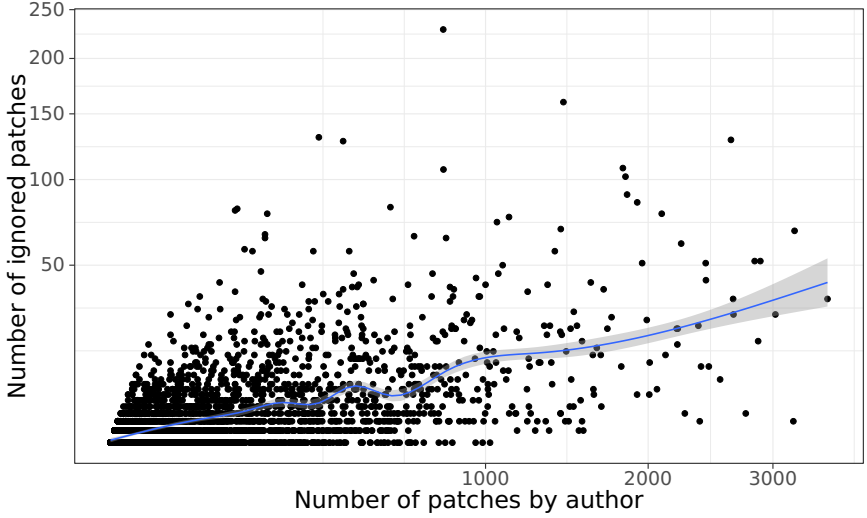
Does it matter *when* a patch is sent?



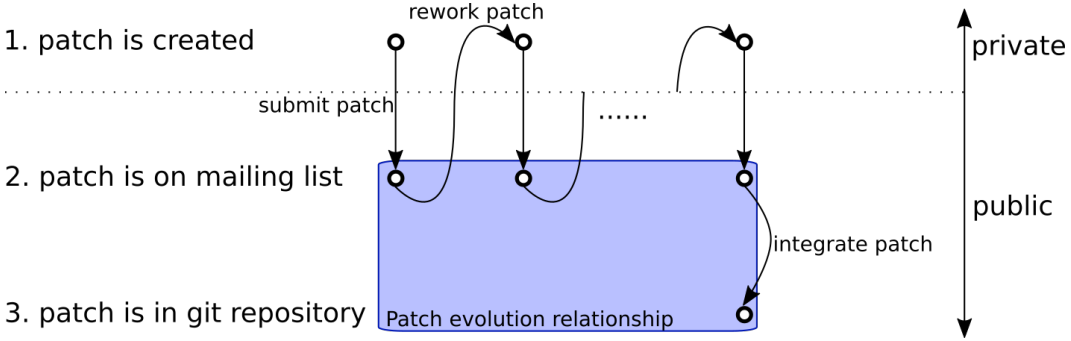
Distribution of ratio ignored/total patches grouped by linux kernel development stage

Insights

- Largely independent of the development stage
- Slightly higher chance of ignorance during merge window


250
200
150
100
50
Number of ignored patches
1000
2000
3000
Number of patches by author

Towards a formal model of the development process



Off-list Patches

Definition

An *off-list patch* is a patch that...

- ...has been included in Linus' git repository
- ...has never been sent to any public mailing list

Off-list Patches

Definition

An *off-list patch* is a patch that...

- ...has been included in Linus' git repository
- ...has never been sent to any public mailing list

Results

- Identified 80 commits with PaStA heuristics from v5.1-rc1..v5.1 (≈1800 commits)
- Manually assessed 60 commits and identified 24 off-list patch commits

Off-list Patches

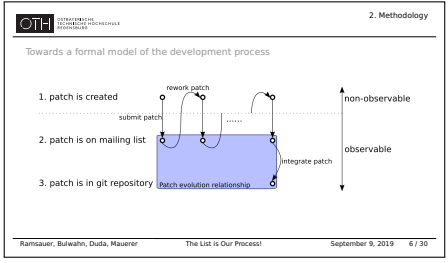
The obvious

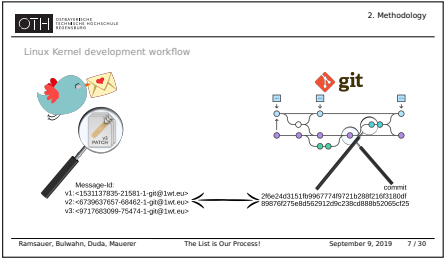
- Reverting patches is discussed on mailing list, the reverting patch is not sent.
- Very few patches from maintainers are actually off-list patches

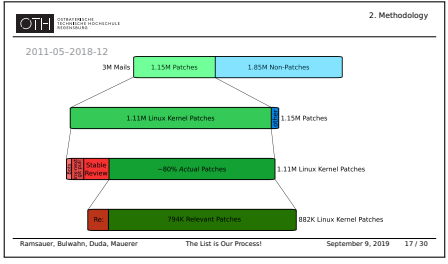
The less obvious

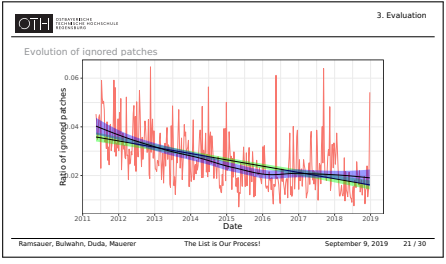
- Some off-list patches are clearly some security-related issues
- Patches from some subsystem maintainers are often off-list

```
commit c7084edc3f6d67750f50d4183134c4fb5712a5c8
Author: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
Date:   Fri Apr 5 15:39:26 2019 +0200

    tty: mark Siemens R3964 line discipline as BROKEN

    The n_r3964 line discipline driver was written in a different time, when
    SMP machines were rare, and users were trusted to do the right thing.
    Since then, the world has moved on but not this code, it has stayed
    rooted in the past with its lovely hand-crafted list structures and
    loads of "interesting" race conditions all over the place.

    After attempting to clean up most of the issues, I just gave up and am
    now marking the driver as BROKEN so that hopefully someone who has this
    hardware will show up out of the woodwork (I know you are out there!)
    and will help with debugging a raft of changes that I had laying around
    for the code, but was too afraid to commit as odds are they would break
    things.

    Many thanks to Jann and Linus for pointing out the initial problems in
    this codebase, as well as many reviews of my attempts to fix the issues.
    It was a case of whack-a-mole, and as you can see, the mole won.

    Reported-by: Jann Horn <jannh@google.com>
    Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
    Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>
```


2. Methodology
Towards a formal model of the development process
1. patch is created
rework patch
submit patch
2. patch is on mailing list
3. patch is in git repository
integrate patch
Patch evolution relationship
non-observable
observable
Ramsauer, Bulwahn, Duda, Mauerer
The List is Our Process!
September 9, 2019
6 / 30


2. Methodology
Linux Kernel development workflow
git
Message-Id:
v1: <1521137835-21581-1-git@1wt.eu>
v2: <67396376 57-68462-1-git@1wt.eu>
v3: <9717683099-75474-1-git@1wt.eu>
commit
2f6e24d3151fb9967774f9721b2880216f3180df
898769275e8d56291209cc238cd888b52005c125
Ramsauer, Bulwahn, Duda, Mauerer
The List is Our Process!
September 9, 2019
7 / 30


2. Methodology
2011-05-2018-12
3M Mails
1.15M Patches
1.85M Non-Patches
1.11M Linux Kernel Patches
1.15M Patches
Stable Review
~80% Actual Patches
1.11M Linux Kernel Patches
Re:
794K Relevant Patches
882K Linux Kernel Patches
Ramsauer, Bulwahn, Duda, Mauerer
The List is Our Process!
September 9, 2019
17 / 30


3. Evaluation
Evolution of ignored patches
Ratio of ignored patches
0.06
0.04
0.02
2011
2012
2013
2014
2015
2016
2017
2018
2019
Date
Ramsauer, Bulwahn, Duda, Mauerer
The List is Our Process!
September 9, 2019
21 / 30

What is useful to measure?

How to improve the process?

BACKUP

What fraction of patches correctly addresses MAINTAINERS

Definition

A patch correctly addresses MAINTAINERS if...

- it addresses **one** ML as listed by get_maintainers.pl
- **OR:** it addresses **one** maintainers as listed by get_maintainers.pl

What fraction of patches correctly addresses MAINTAINERS