Email2git: Matching Linux Code to its Mailing List discussion

alexcourouble.com/oss2017.html

Alexandre Courouble, Bram Adams
Polytechnique Montreal

Kate Stewart
Linux Foundation

Daniel German
University of Victoria
Demo
What is Email2git?

Enter a commit ID

[Input field]

Search

How it works

Email2git is a patch retrieving system built for the Linux kernel.

Enter a git commit ID from the Linux kernel source tree and Email2git will return the patches and discussion associated with the given commit.

Try it! Copy and paste one of the following commit IDs in the search box.

ac401cc782429cc8550ce4840b1405d603740917
b2e0d1625e193b40cbbd45b799f82d54c34e015c
90eec103b96e30401c0b846045bf8a1c7159b6da
Why?

- Linux *contribution process* doesn't provide traceability from *git* to *code review*
  - Mailing list
  - Patchwork
Use cases

• Allows developers to understand design decisions
  • Security
  • Bug fixing
  • New comers
  • <Your use case?>
Use cases

CHA OSS - Community Health Analytics OSS

alexcourouble.com/oss2017.html
Where does the data come from?
The Data

Two Sides

Linux git repository
  git log -p

patchwork.kernel.org
  MySQL queries

@alexcourouble
alexcourouble.com/oss2017.html
Patchwork
Patch tracking system

- Extracts patches from Linux mailing lists
- Keeps track of code reviews
- User friendly interface

alexcourouble.com/oss2017.html @alexcourouble
int __ref create_proc_profile(void) /* false positive from hotcpu_notifier */
{
    struct proc_dir_entry *entry;
    int err = 0;

    if (tprof_on)
        return 0;

    cpu_notifier_register_begin();

    if (create_hash_tables()) {
        err = -ENOMEM;
        goto out;
    }

    entry = proc_create("profile", S_IRUSR | S_IRGRP,
                        NULL, &proc_profile_operations);
    if (!entry)
        goto out;
    proc_set_size(entry, {1 + prof_len} * sizeof(atomic_t));
    __hotcpu_notifier(profile_cpu_callback, 0);

out:
    cpu_notifier_register_done();
    return err;
}

Contributors

<table>
<thead>
<tr>
<th>Contributor</th>
<th>Contributions</th>
<th>Commits</th>
<th>Code Change (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>William Lee Irwin III</td>
<td>38</td>
<td>58.00%</td>
<td></td>
</tr>
<tr>
<td>Srivatsa S. Bhat</td>
<td>26</td>
<td>26.00%</td>
<td></td>
</tr>
<tr>
<td>Paolo Ciarroccoli</td>
<td>5</td>
<td>5.00%</td>
<td></td>
</tr>
<tr>
<td>David Howells</td>
<td>4</td>
<td>4.00%</td>
<td></td>
</tr>
<tr>
<td>Denis V. Lunev</td>
<td>4</td>
<td>4.00%</td>
<td></td>
</tr>
<tr>
<td>Al Viro</td>
<td>2</td>
<td>2.00%</td>
<td></td>
</tr>
<tr>
<td>Dave Hansen</td>
<td>1</td>
<td>1.00%</td>
<td></td>
</tr>
</tbody>
</table>

- Token-level git blame
- ~95% accuracy
- git blame = ~80%
How Does the Algorithm Work?
Will My Patch Make It? And How Fast? 
Case Study on the Linux Kernel

Yujuan Jiang, Brian Adams
McGill University, Canada
{yujuan.jiang, bram.adams}@poly.edu.ca

Daniel M. German
University of Victoria, Canada
dmg@uvic.ca

Abstract—The Linux kernel follows an extremely distributed review protocol, and integration decisions are made within mailing lists and a hierarchy of dozens of Git repositories for version control. Since not every patch can make it and of those that do, some patches require a lot more reviewing and integration effort than others, developers, reviewers and integrators need support for estimating which patches are worthwhile to spend effort on and which ones do not stand a chance. This paper cross- 

ANALYZES AND Merges A 

Eight years of 

Patch reviews 

From the Linux mailing lists and committed patches from the Git repository to understand which patches are accepted and how long it takes those patches to get to the end user. We found that 33% of the patches make it into a Linux release, and that most of them need 3 to 6 months for this. Furthermore, patches developed by more experienced developers are more likely to be accepted and faster reviewed and integrated. Additionally, reviewing time is impacted by submission time, the number of affected subsystems by the patch and the number of reviewed reviewers.

1. INTRODUCTION
Integration of code changes into a project's main repository is an open source developer's ultimate goal, since it marks the first step towards inclusion in an official product release. An open source project like the Linux Kernel, for example, integrates between 8,200 and 12,000 patches in a new release, contributed by more than 1,000 developers [1]. These patches only represent the "lucky few". Studies on Apache and other open source systems have shown that only 40% of the patches considered for integration actually make it into the release [2], [3], [4].

One of the major reasons for the relatively low acceptance rate of integration is the complexity of the process. The patches first need to pass a gate-keeper who performs a review of the code [2], [5], [6], before the code is merged by an integrator (e.g., release engineer) into the corresponding branch of the open source project [7], [8], [9]. Code reviews fail when a patch does not implement a relevant, working feature or bug fix, or when the project's development guidelines are not followed [10]. The actual integration (merging) fails when the patch interacts incorrectly with other patches or the merging process creates too much merge conflicts [11]. In case of integration issues, the developer needs to go back to the drawing board and try to integrate the code again. In the worst case, a patch will be rejected twice and then again until the developer eventually gives up.

As a result, the integration process looks like a black box to most developers, with unpredictable outcome. Everyone knows the stories of disgruntled developers, even experienced ones, whose changes did not make it after putting months of work into the patch (e.g., [12], [13]). Even major projects like the Google Android mobile platform have problems getting their Linux kernel modifications integrated into the official kernel version [1]. Yet, determining upfront whether a patch will make it, and how long it will take, is a gray area. Research on code reviews has shown some small patches [14] and by experienced developers [2] are more likely to be accepted by the reviewers, but it is not clear if these characteristics play the same role during the actual integration of the patch with other patches. Similarly, the impact of these characteristics on the time it takes to get to the patch into a release remains unclear.

This paper studies the relation of patch characteristics with:

1. The probability of acceptance into an official release and
2. The time between submitting a patch for review and acceptance.

We also analyze if these relations change over time. Our empirical analysis is based on eight years of patch review data and version control data from the Linux Kernel project, which is a 20-year-old, popular open source system containing more than 15 million LOC of source code. We address the following research questions:

[RQ1] What percentage of submitted patches has been integra-
ted successfully and how much time did it take?

Around 33% of patches are accepted. Reviewing time has been dropping down to 1-3 months, while integration time steadily has been increasing between 1-3 months, bringing the total time to 3-6 months.

[RQ2] What kind of patch is accepted more likely?

Developer experience, patch maturity and prior sub- system churn play a major role in patch acceptance, while patch characteristics and submission time do not.

[RQ3] What kind of patch is accepted faster?

Reviewing time is impacted by submission time, the number of affected subsystems, the number of suggested reviewers and developer experience, while integration time is impacted by the same attributes as patch acceptance.

First, we provide background about the Linux kernel integration process (Section II), followed by an explanation of our case study methodology (Section III). Section IV presents the results of our case study. Section V is dedicated to a discussion of threats to validity (Section V). We finish the paper with related work (Section VI) and the conclusions (Section VII).

Tacking Back the History of Commits in Low-tech Reviewing Environments

A case study of the Linux kernel

Yujuan Jiang, Brian Adams
McGill University, Canada
{yujuan.jiang, bram.adams}@poly.edu.ca

Foute Khomh
University of Victoria, Canada
foute.khomh@polymtl.ca

Daniel M. German
University of Victoria, Canada
dmg@uvic.ca

ABSTRACT
Context: During software maintenance, people typically go back to the original reviews of a patch to understand the actual design rationale and potential risks of the code. Whereas modern web-based reviewing environments like Gerrit make this process relatively easy, the low-tech, mailing list based reviewing methods of many open source systems make linking a commit back to its reviews and earlier versions far from trivial, since (1) a commit has no physical link with any reviewing email, (2) the discussed patches are not always fully linked to the accepted commits and (3) some discussions last across small multiple threads, each containing potentially multiple versions of the same patch.

Goal: To support maintainers in reconstructing the reviewing history of kernel patches, and studying (for the first time) the characteristics of the reviewed reviewing histories.

Method: This paper performs a comparative empirical study on the Linux kernel mailing lists of 3 small open source projects and email-to-commit linking techniques based on heuristics, commit patch lines and close detection.

Results: Around 25% of the patches had an (until now) hidden reviewing history of more than four weeks, and patches with multiple version typically have a longer and have a higher acceptance rate than patches with just one version.

Conclusion: The patch lines- based linking technique is the best approach for linking patches emails to commits, while it needs to be combined with the checklist-based technique for linking different patch versions.

Categories and Subject Descriptors
D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement—Restructuring, reverse engineering, and reengineering; D.2.9 [Software Engineering]: Management—Software configuration management.

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 ACM 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.

ACM 978-1-5386-2771-9/14/05...

@alexcourouble

alexcourouble.com/oss2017.html
Original Algorithm

Git Log Output

Commit IDs

b3242dba9ff285962fe84d1135cafe9383721f0
67a3b5cb3363339dbb8890ea56c8c1752b541daa
17e34c4fd0be14e989b0873ab302cd357126fe2d
42e6d5e5ee05a0a4f677f1f3eeaf3c367a1498740
105065c3f1f7164d756ee5495b8670e57f2b5f453c
8e793232e1eb6e845b3632a5399c4442de2745
d580e80c7f2f086e241745692ce8bccc91dad
0b31c3ec1b0eac7ca71a7aa4a93d2024cadd33d
1f5d42da44dabd42487ed6d1650eb1c53f87a0

Mailing Lists Patches

Patchwork Patch IDs

...4693
2202
6649
2231
2203
2204
...

Comparing diff’s +/- lines

@alexcourouble
Original Algorithm

Scaling problem

500k + commits since 2009

1.4 M + patches sent since 2009
Scaling the Algorithm

Narrowing down the search space

**Heuristics**

- Email Subject — Commit Summary
- Patch Author
- Files Affected by Patch
Step 1 : Email subject

Git Log

```
commit b2e0d:625e199b40acb0bf517097f82d51d3ead05c
Author: Dan Williams <dan.j.williams@intel.com>
Date: Fri Jan 15 16:55:59 2016 -0800

dax: fix lifetime of in-kernel dax mappings with dax_map_atomic()

The DAX implementation needs to protect new calls to
->direct_access() and usage of its return value against the driver for the
underlying block device being disabled. Use blk_queue_enter()/blk_queue_exit() to
hold off blk_cleanup_queue() from proceeding, or otherwise fail new
mapping requests if the request queue is being torn down.

This also introduces blk_dax_ctl to simplify the interface from fs/dax.c
through dax_map_atomic() to bdev_direct_access().

[willy@linux.intel.com: fix read() of a hole]
Signed-off-by: Dan Williams <dan.j.williams@intel.com>
Reviewed-by: Jeff Moyer <jymoyer@redhat.com>
Cc: Jan Kara <jak@use.com>
Cc: Jens Asbøe <asbob@fb.com>
Cc: Dave Chinner <davchi@sonic.com>
Cc: Ross Zwieler <ross.zwieler@linux.intel.com>
Cc: Matthew Wilcox <willy@linux.intel.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>
```

Linux mailing lists

```
Subject: [PATCH 8/8] dax: fix lifetime of in-kernel dax mappings with
dax_map_atomic()
From: Dan Williams <dan.j.williams@intel.com>
To: linux-nvdimm@lists.01.org
Date: Tue, 17 Nov 2015 12:16:35 -0800
```

This step finds patches for ~55% of commits
Step 2: Author name matching

Git Log

```
commit 7bda66c5e3413ac0495440c41364f35a240f5f55
Author: Dan Williams <dan.j.williams@intel.com>

    dax: fix lifetime of in-kernel dax mappings with dax_map_atomic()

    The DAX implementation needs to protect new calls to __direct_access() and
    usage of its return value against the driver for the underlying block device
    being disabled. Use blk_queue_enter()/blk_queue_exit() to hold off
    blk_createn_queue() from proceeding, or otherwise fail new
    mapping requests if the request_queue is being torn down.

    This also introduces blk_dax_ctl to simplify the interface from fs/dax.c
    through dax_map_atomic() to bdev_direct_access().

[willy@linux.intel: fix read() of a hole]
Signed-off-by: Dan Williams <dan.j.williams@intel.com>
Reviewed-by: Jeff Moyer <moyer@redhat.com>
Cc: Jan Kara <jack@suse.com>
Cc: Jens Axboe <axboe@fb.com>
Cc: Uri Zinner <davidsfromredbit.com>
Cc: Ross Zivkovic <ross.zivkovic@linux.intel.com>
Cc: Matthew Wilcox <willylinux.intel.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>
```

Linux mailing lists

```
Subject: [PATCH 8/8] dax: fix lifetime of in-kernel dax mappings with dax_map_atomic()

From: Dan Williams <dan.j.williams@intel.com>
To: linux-hvdimm@lists.org
Date: Tue, 17 Nov 2015 12:16:35 -0800
```
Step 2: Author name matching

Commit ID: Author Name  ➔  Author Name: Patches sent

“b2e0d16…” ➔  “Dan Williams”  ➔  “Dan Williams” ➔  [Patch 1, Patch 2, Patch 3, …]

Commit - patch matches

+/− line comparison

alexcourouble.com/oss2017.html
Step 3 and 4: File name / file path matching

```
$ git log --name-only --pretty=oneline
```

Patch content

```
diff --git a/include/linux/blkdev.h b/include/linux/blkdev.h
index 3fe27f8d91f0..8aa53454ce27 100644
--- a/include/linux/blkdev.h
+++ b/include/linux/blkdev.h
@@ -1,6 +1,7 @@

+make tlb_flush_pending globals
```

alexcourouble.com/oss2017.html
Step 3 and 4: File name / file path matching

Commit ID : Files touched

“b2e0d16…”
“include/linux/blkdev.h”
“fs/block_dev.c”

→

File name/path : Patches

“fs/dax.c”
“include/linux/blkdev.h”
“fs/block_dev.c”

→

[Patch 1, Patch 2, Patch 3, …]
[Patch 1, Patch 2, Patch 3, …]
[Patch 1, Patch 2, Patch 3, …]

+/− line comparison

Commit - patch matches
How much have we matched?
Percentage of tokens matched in 4.12

@alexcourouble
Mailing lists not tracked yet

net
crypto
security
usr
sound
arch
fs
drivers
ipc
init
scripts
lib
block
samples
mm
kernel
tools
virt

alexcourouble.com/oss2017.html
@alexcourouble
Accessing the Patches
Clicking on a token will fetch the patches for that token.
Email2git interface

Enter a commit ID

b2e0d1625e193b40cbbd45b799f82d54d34e015c

Search

Patch (beta)

Sent: 11/17/2015, 3:16:35 PM

<table>
<thead>
<tr>
<th>#</th>
<th>Patch</th>
<th>Time Sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>linux-block</td>
<td>11/8/2015, 2:27:44 PM</td>
</tr>
<tr>
<td></td>
<td>linux-nwdimm</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>linux-nwdimm</td>
<td>10/9/2015, 8:55:39 PM</td>
</tr>
<tr>
<td></td>
<td>LKML</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>linux-nwdimm</td>
<td>10/22/2015, 1:10:32 PM</td>
</tr>
<tr>
<td></td>
<td>LKML</td>
<td></td>
</tr>
</tbody>
</table>
Limitations

Git

• Only tracking the main linux tree.

• Rebasing

Email

• Currently dependent on Patchwork (only patches sent from 2009)

• Mbox format inconsistent
Future directions

- Rest API to access matches
- Running our own instance of patchwork to parse mailing list archive “in-house” (extracting patch and comments)
- Track linux next
- Improve algorithm for incremental processing
- Patch series and multithreads
- Your feedback

alexcourouble.com/oss2017.html