

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

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.

ac401cc782429cc8560ce4840b1405d503740917
b2e0d1625e193b40cbbd45b799f82d54d34e015c
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

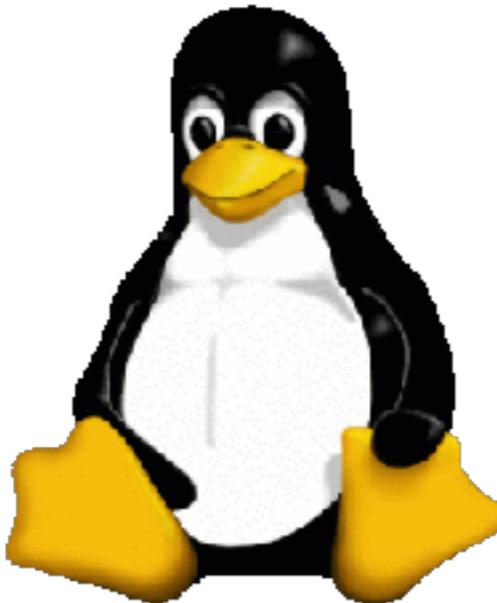


CHAOSS - Community Health Analytics OSS

Where does the data come from?

The Data

Two Sides



Linux git repository

`git log -p`



patchwork.kernel.org

MySQL queries

Patchwork

Patch tracking system



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



```
int __ref create_proc_profile(void) /* false positive from hotcpu_notifier */
{
    struct proc_dir_entry *entry;
    int err = 0;

    if (!prof_on)
        return 0;

    cpu_notifier_register_begin();

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

    entry = proc_create("profile", S_IWUSR | S_IRUGO,
                       NULL, sproc_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

william lee irwin iii	william lee irwin iii	58	58.00%
sriivatsa s. bhat	sriivatsa s. bhat	26	26.00%
paolo ciarrocchi	paolo ciarrocchi	5	5.00%
david howells	david howells	4	4.00%
denis v. lunev	denis v. lunev	4	4.00%
al viro	al viro	2	2.00%
dave hansen	dave hansen	1	1.00%

- 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, Bram Adams
MCIS, Polytechnique Montréal, Canada
{yujuan.jiang,bram.adams}@polymtl.ca

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

Abstract—The Linux kernel follows an extremely distributed reviewing and integration process supported by 130 developer 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-links and analyzes eight years of patch reviews from the kernel 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 makes it into a Linux release, and that most of them need 3 to 6 months for this. Furthermore, that patches developed by more experienced developers are more easily 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 requested reviewers.

I. 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 8,000 and 12,000 patches in a new release, contributed by more than 1,000 developers [1]. Those patches only represent the "lucky few". Studies on Apache and other open source systems have shown how only 40% of the patches considered for integration eventually succeed [2], [3], [4].

One of the major reasons for the relatively low success rate of integration is the complexity of this 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 many 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 time and time 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 them (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 [11]. Yet, determining up front whether a patch will make it, and how long it will take, is a grey area. Research on code reviews has shown how small patches [6], [4] sent 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 a patch into a release is 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 MLOC of source code. We address the following research questions:

RQ1) What percentage of submitted patches has been integrated 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 towards 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 subsystem 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, followed by a discussion of threats to validity (Section V). We finish the paper with related work (Section VI) and the conclusion (Section VII).

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

A case study of the Linux kernel

Yujuan Jiang,
Bram Adams
MCIS, Polytechnique
Montréal, Canada
{yujuan.jiang,bram.adams}@polymtl.ca

Foulse Khomh
SWAT, Polytechnique
Montréal, Canada
foulse.khomh@polymtl.ca

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

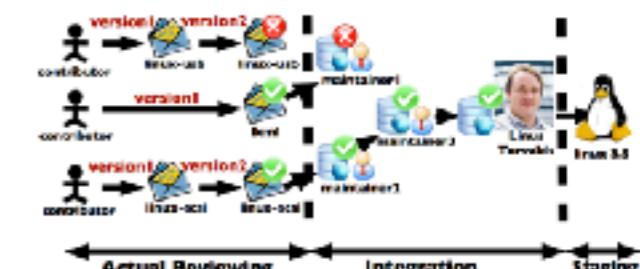


Figure 1: The reviewing and integration process of a patch in the Linux kernel project (adapted from [19]).

General Terms

Software Engineering

Keywords

review, traceability, clone detection, low-tech reviewing environment, mailing list, open source, Linux kernel

I. INTRODUCTION

In November 2003, there was a suspected backdoor attempt to taint the source code of the Linux kernel project [23]. The Linux team noticed that a patch popped up in the CVS copy of the version control system that never appeared in the BitKeeper master copy. This patch pretended to just check for errors, but actually contained a back door that could render a system vulnerable. Up until today, it is still not clear whether this code was inserted by a malicious hacker, since there was (and still is) no explicit link from a commit back to all emails reviewing that patch or earlier versions of it. The recent "heartbleed" vulnerability [16] also highlights the need to have traceability from commits in the version control system to patches in the mailing list. Full traceability would have made a quicker and more accurate audit of the code possible. Apart from security audits, traceability from submitted patches to accepted commits would also benefit regular maintenance, as it will make it easy to review the email messages around that patch which comprise the reviewing and design discussions of the patches [2][3][26].

Although more and more open and closed source systems are migrating towards modern, online reviewing environments like Gerrit, which consolidate all patch versions, dis-

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. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ESEM '14 September 18–19, 2014, Torino, Italy.

Copyright 2014 ACM 978-1-4503-2774-9/14/09...\$15.00.

Original Algorithm

Git Log Output

Commit IDs
...
b3242dba9ff285962fe84d1135cafe9383d721f0
67a3b5cb33633f39db8809ae56c8c1752b541daa
17e34c4fd0be14e989b08734b302cd357126fe2d
42e6d5e5ee05a0a4fa6f71ffeeafc367a1483740
105065c3f7164d756ee5495b8670e57f2bcf65c3
8c7932a32e1ebf6e845bd32a5399cce442d2e745
d580e80c7f2ffb8f86e2417495692ce8bcc91dad
0b31c3ec1b0eac7ca71a7aa4a4a93d02fcade33d
1f5de42da44adbd4248f7ed6d1650eb1c53f87a0
...

```
50 51     -int dax_clear_blocks(struct dm_ndnode *ndnode, sector_t block, long size)
52 53     +int dax_clear_blocks(struct dm_ndnode *ndnode, sector_t block, long _size);
54 55     {
56 56         struct block_device *bdev = ndnode->dm->dev;
57 57         sector_t sector = block << (ndnode->dm_blnbits - 9);
58 58         struct blk_dax_ctl *dax =
59 59             sector + block << (ndnode->dm_blnbits - 9),
60 60             .size = _size,
61 61         );
62 62         eight_align();
63 63     }
64 64     void __iomem *addr;
65 65     unsigned long pfn;
66 66     long count, sz;
67 67     count = bdev_direct_access(ndnode, sector, &addr, &pfn, size);
68 68     if (count < 0)
69 69         return count;
70 70     if (size <= SZ)
71 71         clear_pmem(addr, sz);
72 72     else
73 73         clear_pmem(dax->addr, sz);
74 74     dax.size -= sz;
75 75     dax.sector += sz / SZ;
76 76     dax_sectors_update(ndnode, &dax);
77 77 }
```

Mailing Lists Patches

Patchwork Patch IDs

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

```
diff --git a/block/blk.h b/block/blk.h
index da722eb786df..c43926d3d74d 100644
--- a/block/blk.h
+++ b/block/blk.h
@@ -92,6 +92,6 @@ void blk_dequeue_request(struct request *rq);
 void blk_queue_free_tags(struct request_queue *q);
 bool blk_end bidi_request(struct request *rq, int error,
                           unsigned int nr_bytes, unsigned int bidi_bytes);
-int blk_queue_enter(struct request_queue *q, gfp_t gfp);
+void blk_queue_exit(struct request_queue *q);
 void blk_freeze_queue(struct request_queue *q);

 static inline void blk_queue_enter_llow(struct request_queue *q)
diff --git a/fs/block_dev.c b/fs/block_dev.c
index e2cc681ddafe..228c313c0ac1 100644
--- a/fs/block_dev.c
+++ b/fs/block_dev.c
@@ -436,10 +436,7 @@ EXPORT_SYMBOL_GPL(bdev_write_page);
 /**
  * bdev_direct_access() - Get the address for directly-accessible memory
  * @bdev: The device containing the memory
  * @sector: The offset within the device
  * @addr: Where to put the address of the memory
  * @pfn: The Page Frame Number for the memory
  * @sz: The number of bytes requested
+ * @dma: Control and output parameters for ->direct_access
  *
  * If a block device is made up of directly addressable memory, this function
  * will tell the caller the PFN and the address of the memory. The address
@@ -450,10 +447,10 @@ EXPORT_SYMBOL_GPL(bdev_write_page);
  * returns negative errno if an error occurs, otherwise the number of bytes
  * accessible at this address.
  */
-long bdev_direct_access(struct block_device *bdev, sector_t sector,
-                      void __iomem **addr, unsigned long *pfn, long size)
+long bdev_direct_access(struct block_device *bdev, struct blk_dax_ctl *dax)
{
    long avail;
+    sector_t sector = dax->sector;
+    long avail, size = dax->size;
    const struct block_device_operations *ops = bdev->bd_disk->ops;
```

Comparing diff's
+/- lines

Original Algorithm

Scaling problem

500k +
commits since
2009

1.4 M + patches
sent since 2009

Scaling the Algorithm

Narrowing down the search space

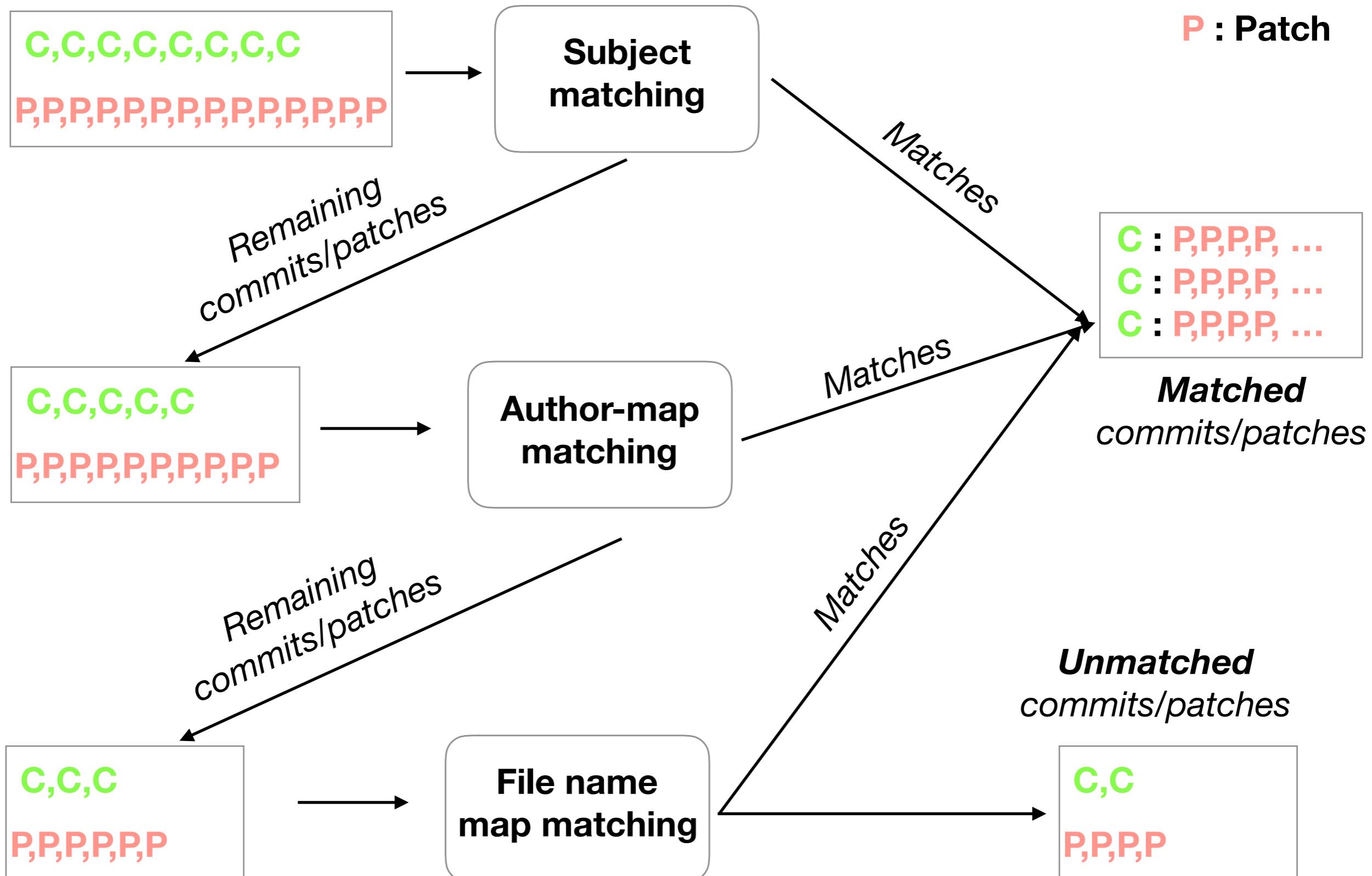
Heuristics

Email Subject – Commit Summary

Patch Author

Files Affected by Patch

C : Commit
P : Patch



Step 1 : Email subject

Git Log

```
commit b2e0d1625e193b40cbbd45b799f82d54d34e015c
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 <jmoyer@redhat.com>
Cc: Jan Kara <jack@suse.com>
Cc: Jens Axboe <axboe@fb.com>
Cc: Dave Chinner <davidgfromorbit.com>
Cc: Ross Zwisler <ross.zwisler@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

```
Author: Dan Williams <dan.j.williams@intel.com>
Date: Tue Nov 17 12:16:35 -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 <jmoyer@redhat.com>
Cc: Jan Kara <jack@suse.com>
Cc: Jens Axboe <axboe@fb.com>
Cc: Dave Chinner <davidgfromorbit.com>
Cc: Ross Zwisler <ross.zwisler@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-nvme@lists.vi.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, ...]

```
35 -int dax_clear_blocks(struct inode *inode, sector_t block, long size)
36 +int dax_clear_blocks(struct inode *inode, sector_t block, long size)
37 38 {
39 40     struct block_device *bdev = inode->i_sb->sb_bdev;
41 -    sector_t sector = block << (inode->i_blksize - 9);
42 +    struct blk_dax_ctl dax = {
43 +        .sector = block << (inode->i_blksize - 9),
44 +        .size = _SIZP,
45 +    };
46 47     might_sleep();
47     do {
48 -        void __pmem *addr;
49 -        unsigned long pfs;
50 51         long count, sz;
52 53         count = bdev_direct_access(bdev, sector, &addr, &pfs, size);
53 54         count = dax_map_atomic(bdev, &dax);
55 56         if (count < 0)
56 57             return count;
57 58         sz = min_t(long, count, SZ_128K);
58 59         clear_pmem(addr, sz);
59 60         size -= sz;
60 61         sector += sz / 512;
61 62         clear_pmem(dax.addr, sz);
62 63         dax.size -= sz;
63 64         dax.sector += sz / 512;
64 65         dax_unmap_atomic(bdev, &dax);
```

→ +/- line comparison

↓
Commit - patch matches

Step 3 and 4: File name / file path matching

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

```
00baec21e4585f4258f010582c5e23f1e5edc98c mm: fix MADV_{FREE|DONTNEED} TLB flush miss problem
arch/arm/include/asm/tlb.h
arch/ia64/include/asm/tlb.h
arch/s390/include/asm/tlb.h
arch/sh/include/asm/tlb.h
arch/um/include/asm/tlb.h
include/asm-generic/tlb.h
include/linux/mm_types.h
mm/memory.c
ea2d266dd0733150930000000000000061a6b446d mm: make tlb_flush_pending global
include/linux/mm_types.h
mm/debug.c
56236a59556cf03bae7bffb7e5f433b5cf0af880 mm: refactor TLB gathering API
arch/arm/include/asm/tlb.h
arch/ia64/include/asm/tlb.h
arch/s390/include/asm/tlb.h
arch/sh/include/asm/tlb.h
arch/um/include/asm/tlb.h
include/asm-generic/tlb.h
include/linux/mm_types.h
mm/memory.c
```

Patch content

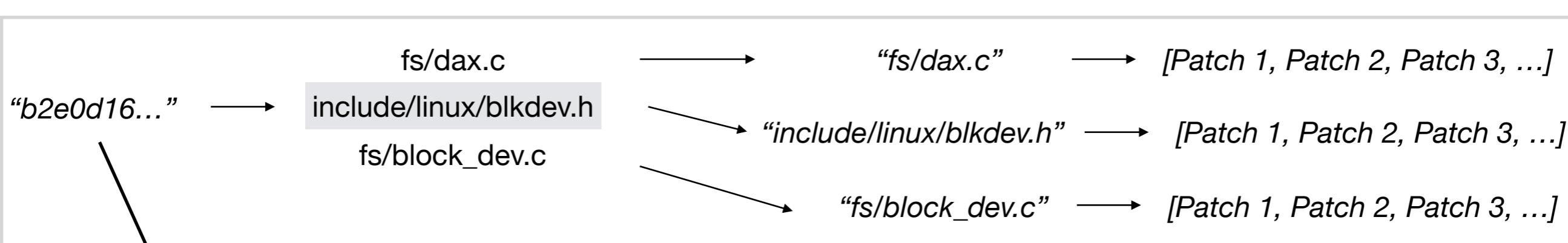
```
diff --git a/include/linux/blkdev.h b/include/linux/blkdev.h
index 3fe27f8d91f0..8aa53454ce27 100644
similarity token 100%
+++ b/include/linux/blkdev.h
```

Step 3 and 4: File name / file path matching

Commit ID : Files touched



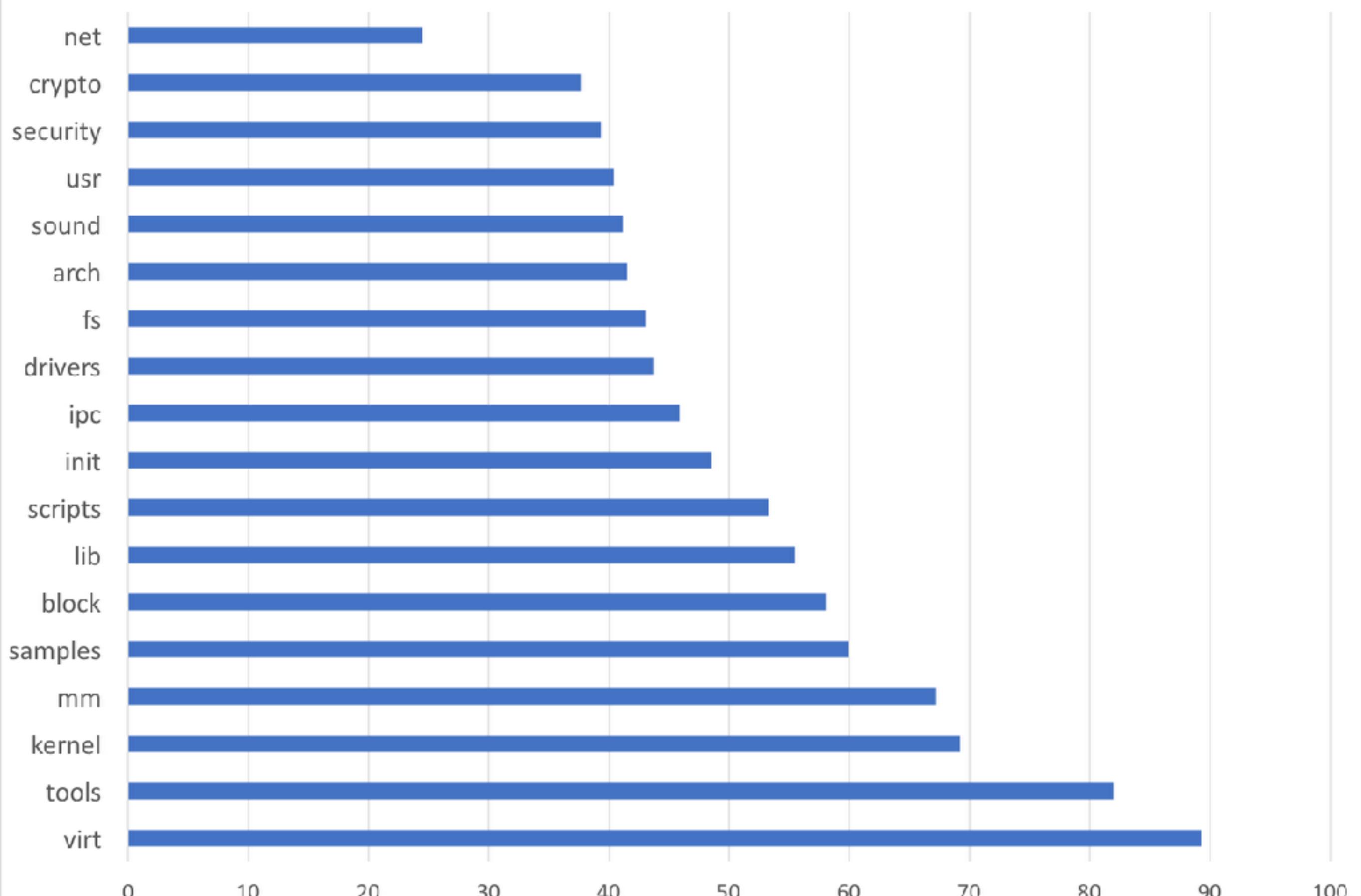
File name/path : Patches



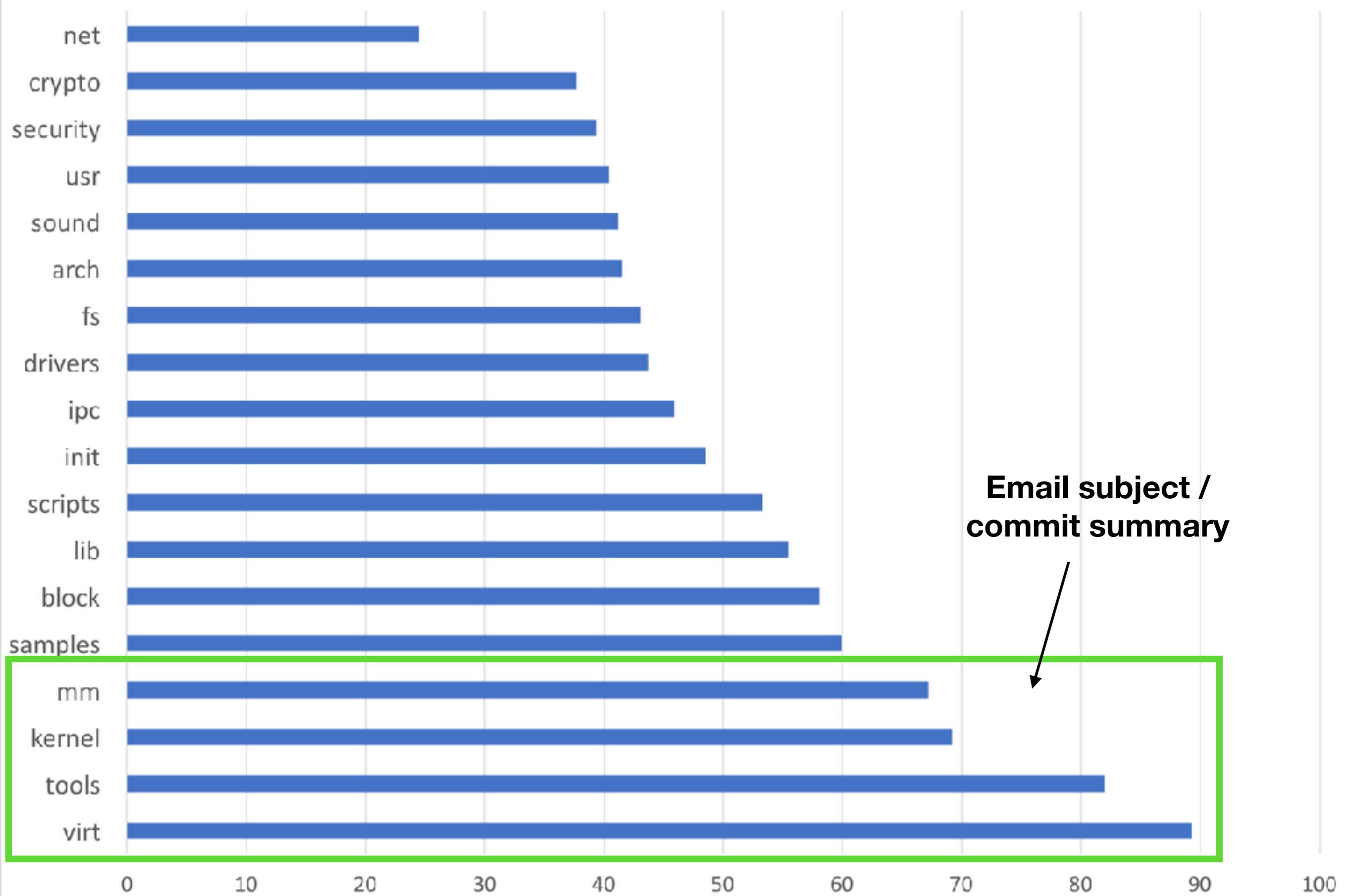
```
38 -int dax_clear_block(struct inode *inode, sector_t block, long size)
39 +int dax_clear_blocks(struct inode *inode, sector_t block, long size)
40 41 {
41 42     struct block_device *bdev = inode->i_sb->sb_bdev;
42 43     sector_t sector = block << (inode->i_blkbits - 9);
43 44     struct blk_dax_ctl dax = {
44 45         .sector = sector << (inode->i_blkbits - 9),
45 46         .size = _S128,
46 47     };
47 48
48 49     might_sleep();
49 50     do {
50 51         void __pmem *addr;
51 52         unsigned long pfn;
52 53         long count, sz;
53 54
54 55         count = bdev_direct_access(bdev, sector, &addr, &pfn, size);
55 56         if (count < 0)
56 57             return count;
57 58         sz = min_t(long, count, SZ_128K);
58 59         clear_pmem(addr, sz);
59 60         size -= sz;
60 61         sector += sz / S12;
61 62         clear_pmem(dax.addr, sz);
62 63         dax.size -= sz;
63 64         dax.sector += sz / S12;
64 65         dax_unmap_atomic(bdev, &dax);
65 66 }
```

+/- line comparison

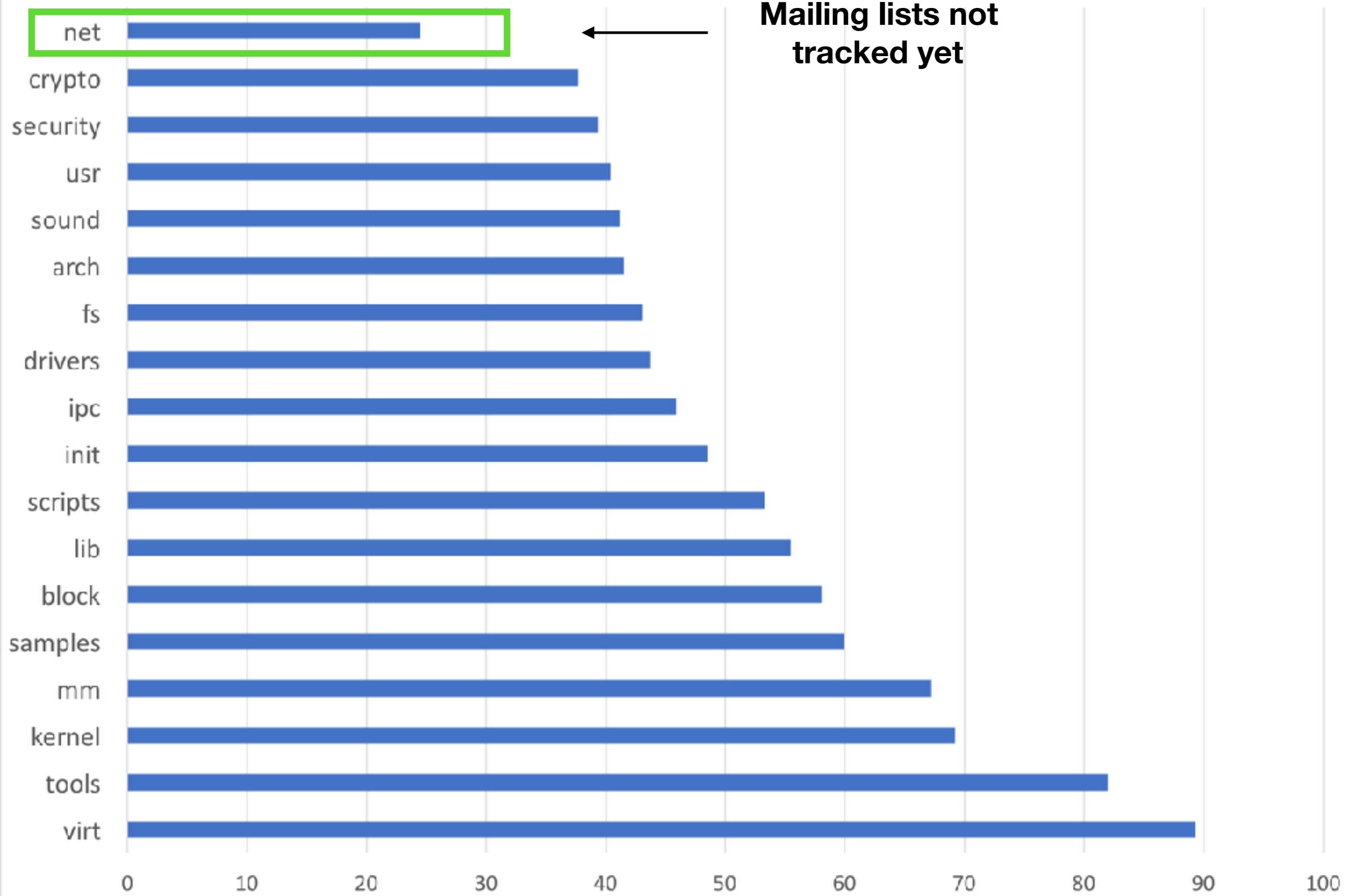
How much have we matched?



Percentage of tokens matched in 4.12



Email subject /
commit summary



Accessing the Patches



1: (sig_len >= modlen)

3f1e1bea34740069f70c6bc92d0f712345d5c28e

[View commit on github](#) [View file on LXR](#)

Patch (beta)

LKML

Sent: 11/26/2014, 9:18:04 AM

Patch Discussion / Previous Attempts

#	Patch	Time Sent
1	LKML	9/8/2014, 11:39:24 AM
2	LKML linux wireless	5/28/2015, 11:46:52 AM
3	LKML	8/5/2015, 9:44:37 AM
4	LKML	2/6/2015, 9:59:24 AM
5	LKML	10/3/2014, 10:30:40 AM
6	linux-wireless LKML	7/17/2015, 12:15:56 PM
7	LKML	5/15/2015, 8:36:00 AM
8	LKML	11/20/2014, 11:54:48 AM

Clicking on a token will fetch the patches for that token

Email2git interface

Enter a commit ID

Search



Patch

Patch (beta)

linux-nvdimm | linux-block | linux-fsdevel

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

Patch Discussion / Previous Attempts

#	Patch	Time Sent
1	linux-block linux-nvdimm	11/8/2015, 2:27:44 PM
2	linux-nvdimm LKML	10/9/2015, 8:55:39 PM
3	linux-nvdimm LKML	10/22/2015, 1:10:32 PM

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