Kernel documentation: what we have and where we're going

Jonathan Corbet
corbet@lwn.net

Why does documentation matter?

A crucial aid to our users

An aid for our developers

It makes us think about what we’re doing

Documentation is a key to building a healthy community

The Linux kernel

The core of any Linux system

Some numbers:

- 68,000 files
- 5,000 directories
- 63-70 day release cycle (+/-)
- 1,700 developers contributing to each release
  (>4000 over the course of a year)
- 13,000 changesets (at least) in each release

A huge and fast-moving project!

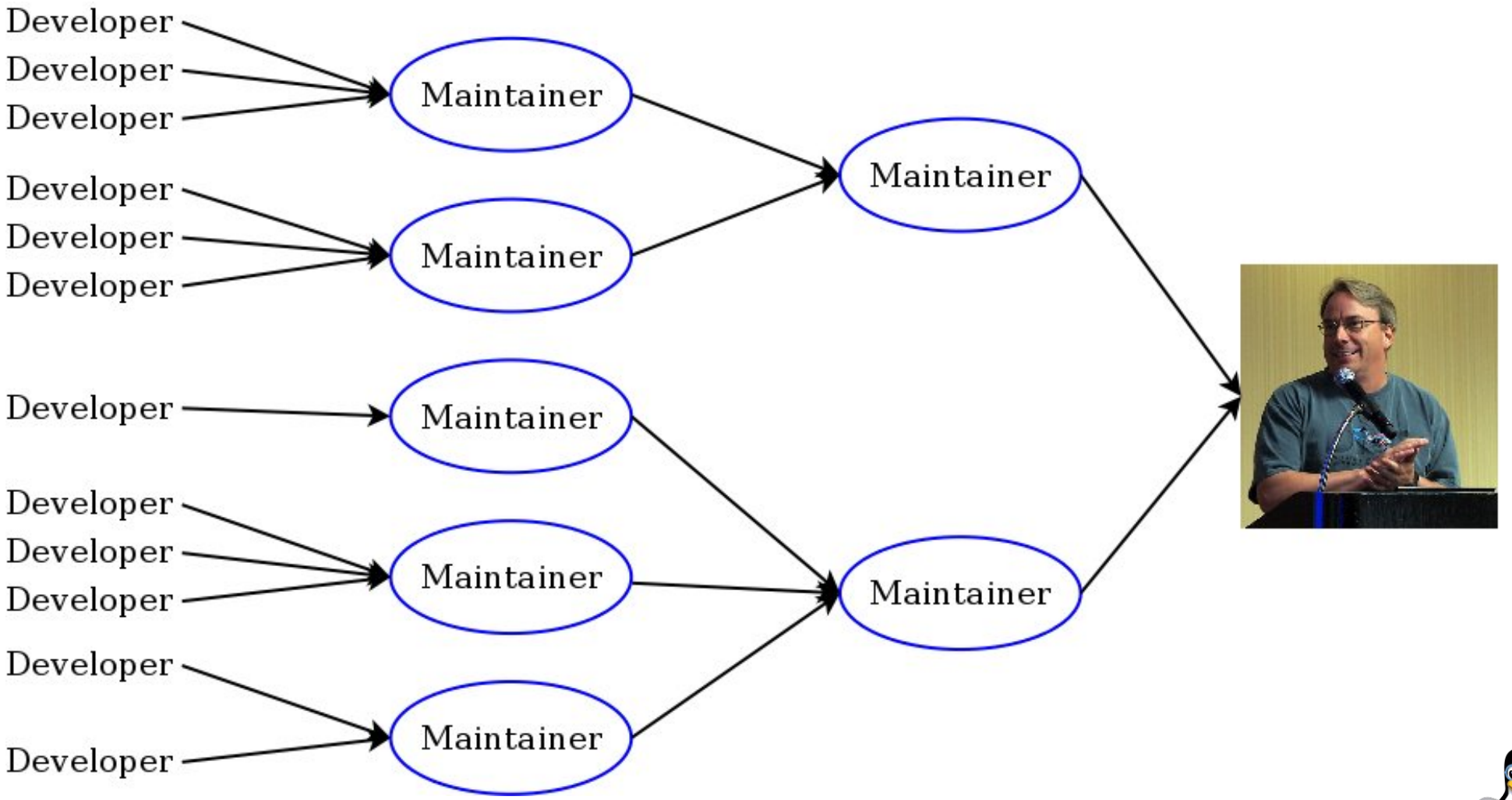
Interesting kernel facts

90% (or more) of kernel code is written by paid developers

Nobody is paid to write kernel documentation

Interesting kernel facts

The kernel has a well-defined maintainer model


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

The maintainer model

...closely matches the kernel file hierarchy

The SCSI maintainer manages drivers/scsi/

Documentation does not fit this model

Everybody touches Documentation/

Lots of documentation lives elsewhere

Kernel developers are conservative

The end result

Just being the docs maintainer is an interesting challenge

The end result

> Documentation/* is a gigantic mess, currently organized based on where random passers-by put things down last.
> — Rob Landley, July 2007

Kernel documentation in 2016

Over 2,000 .txt files

34 DocBook “template files”

Thousands of kerneldoc comments in source

Kerneldoc comments

Found throughout the kernel source

```
/**
 * list_add - add a new entry
 * @new: new entry to be added
 * @head: list head to add it after
 *
 * Insert a new entry after the specified head.
 * This is good for implementing stacks.
 */
```

2016: What's not to like?

A fragile, complex, home-made build system

No markup in kerneldoc comments

2,000 standalone bits of text

An unpleasant experience for everybody involved

Something happened in 4.8

DocBook replaced with Sphinx

- Documentation formatted with RestructuredText

Rebasing
========

"Rebasing" is the process of changing the history of a series of commits within a repository. There are two different types of operations that are referred to as rebasing since both are done with the ``git rebase`` command, but there are significant differences between them:

- Changing the parent (starting) commit upon which a series of patches is built. For example, a rebase operation could take a patch set built on the previous kernel release and base it, instead, on the current release. We'll call this operation "reparenting" in the discussion below.

- Changing the history of a set of patches by fixing (or deleting) broken commits, adding patches, adding tags to commit changelogs, or changing the order in which commits are applied. In the following text, this type of operation will be referred to as "history modification"

The term "rebasing" will be used to refer to both of the above operations.

Something happened in 4.8

- DocBook replaced with Sphinx
  - Documentation formatted with RestructuredText
- Kerneldoc comments can use RST

Something happened in 4.8

- DocBook replaced with Sphinx
  - Documentation formatted with RestructuredText
- Kerneldoc comments can use RST
- Old toolchain thrown away

> The merge window has been fairly normal, although the patch itself looks somewhat unusual: over 20% of the patch is documentation updates, due to conversion of the drm and media documentation from docbook to the Sphinx doc format.
> — Linus Torvalds (4.8-rc1 release)

What we were trying to do

Easy, plain-text formatting

Create an integrated set of kernel documents

Preserve readability of plain-text documentation

Encourage the creation of more docs

The current state of kernel documentation

In `Documentation/`

3,054 files

(excluding `Documentation/devicetree`)

2,322 in 4.7

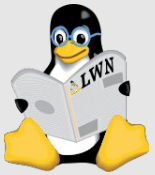
In `Documentation/`

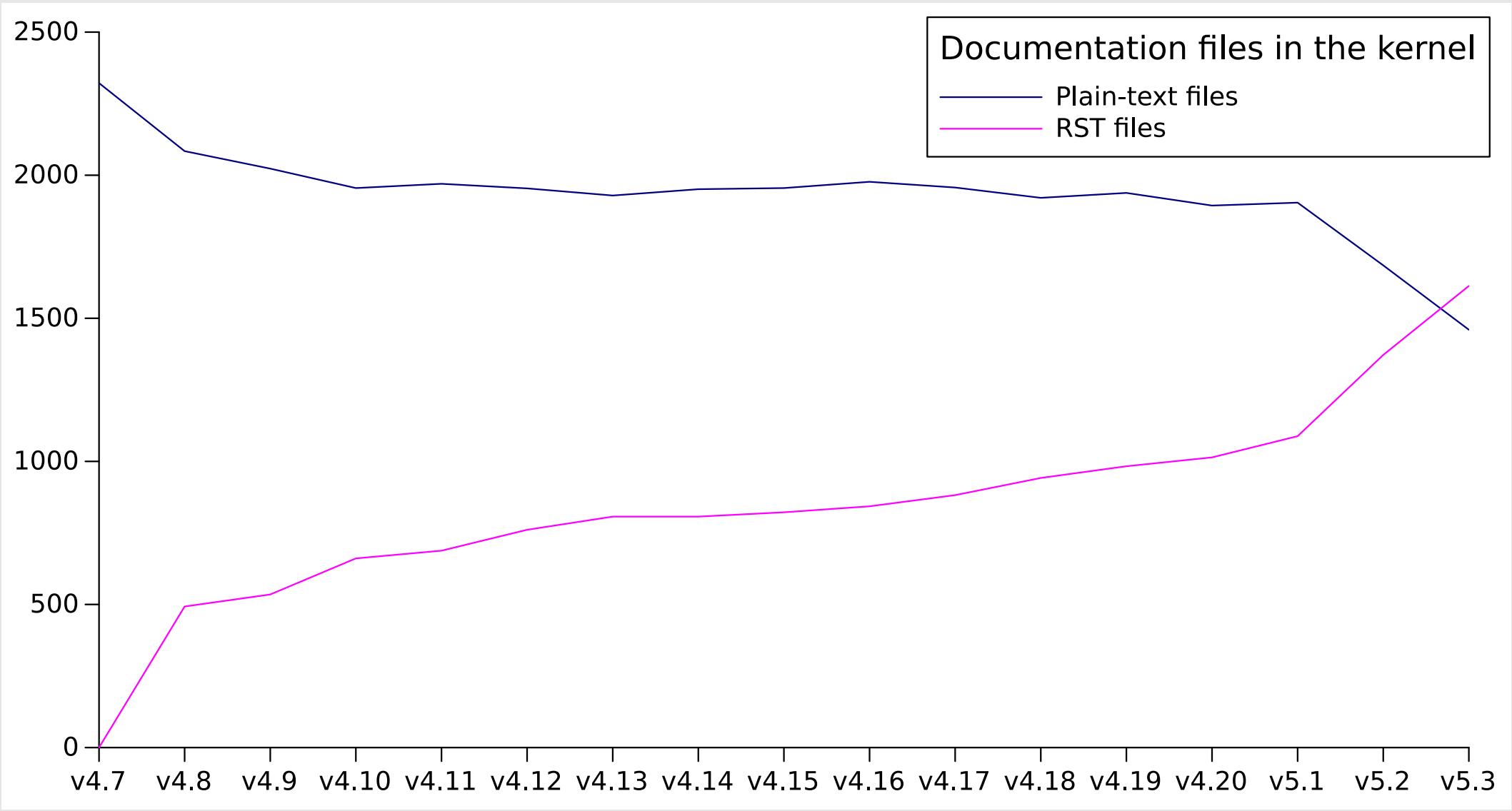
3,054 files

(excluding `Documentation/devicetree`)

2,322 in 4.7

1,536 `.rst` files




Documentation files in the kernel
Plain-text files
RST files
2500
2000
1500
1000
500
0
v4.7
v4.8
v4.9
v4.10
v4.11
v4.12
v4.13
v4.14
v4.15
v4.16
v4.17
v4.18
v4.19
v4.20
v5.1
v5.2
v5.3

Elsewhere

A vast and growing collection of kerneldoc comments

Kerneldoc comments

Found throughout the kernel source

```
/**
 * list_add - add a new entry
 * @new: new entry to be added
 * @head: list head to add it after
 *
 * Insert a new entry after the specified head.
 * This is good for implementing stacks.
 */
```

The Linux Kernel API

List Management Functions

void list_add(struct list_head * *new*, struct list_head * *head*)

add a new entry

Parameters

`struct list_head * new`

new entry to be added

`struct list_head * head`

list head to add it after

Description

Insert a new entry after the specified head. This is good for implementing stacks.

void list_add_tail(struct list_head * *new*, struct list_head * *head*)

add a new entry

Parameters

`struct list_head * new`

```
/**
 * DOC: dma buf device access
 *
 * For device DMA access to a shared DMA buffer the usual sequence of operations
 * is fairly simple:
 *
 * 1. The exporter defines his exporter instance using
 *    DEFINE_DMA_BUF_EXPORT_INFO() and calls dma_buf_export() to wrap a private
 *    buffer object into a &dma_buf. It then exports that &dma_buf to userspace
 *    as a file descriptor by calling dma_buf_fd().
 *
 * 2. Userspace passes this file-descriptors to all drivers it wants this buffer
 *    to share with: First the filedescriptor is converted to a &dma_buf using
 *    dma_buf_get(). Then the buffer is attached to the device using
 *    dma_buf_attach().
```

Basic Operation and Device DMA Access

~~~~~~

```
.. kernel-doc:: drivers/dma-buf/dma-buf.c
   :doc: dma buf device access
```


~~~~~~

- The DMA buffer FD is also pollable, see Fence Poll Support below for details.

Basic Operation and Device DMA Access

For device DMA access to a shared DMA buffer the usual sequence of operations is fairly simple:

1. The exporter defines his exporter instance using `DEFINE_DMA_BUF_EXPORT_INFO()` and calls `dma_buf_export()` to wrap a private buffer object into a `dma_buf`. It then exports that `dma_buf` to userspace as a file descriptor by calling `dma_buf_fd()`.
2. Userspace passes this file-descriptors to all drivers it wants this buffer to share with: First the filedescriptor is converted to a `dma_buf` using `dma_buf_get()`. Then the buffer is attached to the device using `dma_buf_attach()`.

   Up to this stage the exporter is still free to migrate or reallocate the backing storage.
3. Once the buffer is attached to all devices userspace can initiate DMA access to the shared buffer. In the kernel this is done by calling `dma_buf_map_attachment()` and `dma_buf_unmap_attachment()`.
4. Once a driver is done with a shared buffer it needs to call `dma_buf_detach()` (after cleaning up any mappings) and then release the reference acquired with dma_buf_get by calling `dma_buf_put()`.

For the detailed semantics exporters are expected to implement see `dma_buf_ops`.

CPU Access to DMA Buffer Objects

There are mutliple reasons for supporting CPU access to a dma buffer object:

- Fallback operations in the kernel, for example when a device is connected over USB and the kernel needs to shuffle the data around first before sending it away. Cache coherency is handled by braketing any transactions with calls to `dma_buf_begin_cpu_access()` and `dma_buf_end_cpu_access()` access.

  To support dma_buf objects residing in highmem cpu access is page-based using an api similar to kmap. Accessing a dma_buf is done in aligned chunks of PAGE_SIZE size. Before accessing a chunk it needs to be mapped, which returns a pointer in kernel virtual address space. Afterwards the chunk needs to be unmapped again. There is no limit on how often a given chunk can be mapped and unmapped, i.e. the importer does not

We have come a long way!

My general impression is that it is now a way easier to maintain the media documentation and make it more consistent than with DocBook.
— Mauro Carvalho Chehab

This new documentation format combines the best of two worlds, pretty online browser documentation with almost plain text files, and changes being tracked via git commits.... You got to love it! :-)
— Jesper Dangaard Brouer

What’s next?

Build warnings

```
./include/linux/netdevice.h:2040: warning:
Function parameter or member 'xps_rxqs_map' not
described in 'net_device'

./include/linux/xarray.h:232: WARNING: Unexpected
indentation.
```

Convert the remaining .txt files

...in progress...

Ancient documents

Ancient documents

Documentation/platform/x86-laptop-drivers.txt

```
compal-laptop
=============
List of supported hardware:

by Compal:
    Compal FL90/IFL90
    Compal FL91/IFL91
    Compal FL92/JFL92
    Compal FT00/IFT00

by Dell:
    Dell Vostro 1200
    Dell Mini 9 (Inspiron 910)
    Dell Mini 10 (Inspiron 1010)
    Dell Mini 10v (Inspiron 1011)
    Dell Mini 1012 (Inspiron 1012)
    Dell Inspiron 11z (Inspiron 1110)
    Dell Mini 12 (Inspiron 1210)
```

Converting documents to RST?

easy!

Converting documents to RST?

easy!

Evaluating for relevance and correctness?

Updating them to match reality?

...less so

Organization

Documentation is for the readers

Kernel documentation “books”

core-api/
: Core kernel API stuff

userspace-api/
: Stuff for application developers

process/
: How to participate in kernel development

admin-guide/
: Stuff for sysadmins

dev-tools/
: Tools for kernel development

...

Integration

> Hmmm...probably premature to bring this up, but Documentation/dev-tools/ is kind of thrown together.
> — Brendan Higgins

Integration

The kernel-doc mechanism is nice, but...

- It does split documents across files

Missing manuals

Maintainers guide

Subsystem guides for developers

...

Toolchain improvements

- scripts/kernel-doc
  - 2200 lines of ancient Perl
- PDF generation
  - Still depends on LaTeX
  - Fragile
- Sphinx stylesheets
  - ugly!

Win over doubters

> I don't much care for Documentation/ -- code should be readable and have sufficient comments; I hate rst and I think that anything that detracts from reading code comments in an editor is pure evil.
> — Peter Zijlstra

Winning over doubters

Sphinx has a syntax for function references:

```
:c:func:``kmalloc()``
```

Winning over doubters

Sphinx has a syntax for function references:

```
:c:func:``kmalloc()``
```

Automarkup extension added in 5.3

Just write `kmalloc()`

Write more documentation!

If you want to be a part of kernel development

...please consider working on documentation

Questions / thoughts?