Driver Linux (DRV)

How to upstream

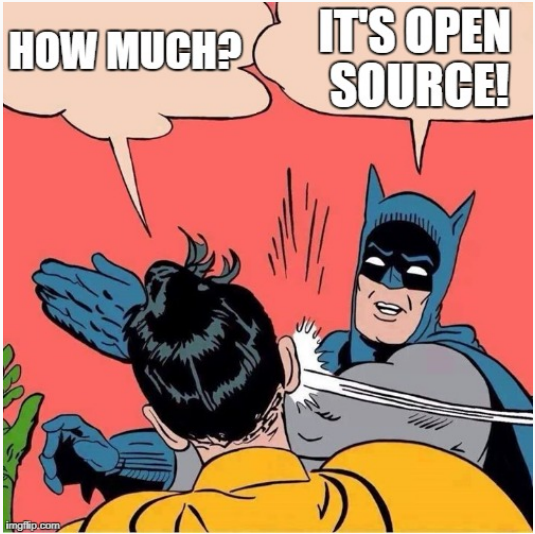
Florian Vaussard

Juin 2024

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Introduction

What is Open-Source?



What is Open-Source?

Wikipedia:

> *Open source is a source code that is made freely available for possible modification and redistribution.*

Some related (philosophical) concepts:

- *Free Software*, *FOSS*, etc.
- See Richard Stallman:
  https://www.gnu.org/philosophy/free-software-for-freedom.html

What is Open-Source?

Core concepts

- Publicly accessible
- Able to share with everyone
- Able to inspect, modify, enhance

Some misconceptions:

- Done by unpaid developers
- Less secure than proprietary software
- More secure than proprietary software

Unpaid developers?

Most active 6.9 employers

By changesets			By lines changed		
Intel	1867	12.9%	AMD	171877	21.7%
(Unknown)	1072	7.4%	Red Hat	91448	11.5%
Google	1031	7.1%	Intel	70800	8.9%
(None)	979	6.8%	Google	51104	6.5%
Linaro	924	6.4%	Oracle	47906	6.0%
AMD	820	5.7%	(Unknown)	44300	5.6%
Red Hat	807	5.6%	Linaro	41492	5.2%
SUSE	468	3.2%	(None)	28388	3.6%
Meta	413	2.9%	Qualcomm	17812	2.2%
Pengutronix	372	2.6%	Meta	17388	2.2%
Huawei Technologies	345	2.4%	Renesas Electronics	17051	2.2%
Oracle	313	2.2%	Realtek	13862	1.7%
Qualcomm	311	2.1%	SUSE	11953	1.5%
IBM	301	2.1%	NVIDIA	10162	1.3%
(Consultant)	287	2.0%	Huawei Technologies	9100	1.1%
Renesas Electronics	247	1.7%	(Consultant)	7140	0.9%
NVIDIA	241	1.7%	IBM	6777	0.9%
Texas Instruments	210	1.5%	Collabora	6760	0.9%
Arm	176	1.2%	Arm	6712	0.8%
Microsoft	159	1.1%	Marvell	6587	0.8%

Some 6.9 development statistics: `https://lwn.net/Articles/972605/`

Open-Source Licenses

(Too) Many different licenses:

- GPL (v2/v3)
- LGPL
- BSD
- MIT
- JSON ("The Software shall be used for Good, not Evil.")
- ...

Beware:

- Linking with a library (e.g. GPL versus LGPL)
- Copyleft issues
- Patents, etc.

But IANAL...

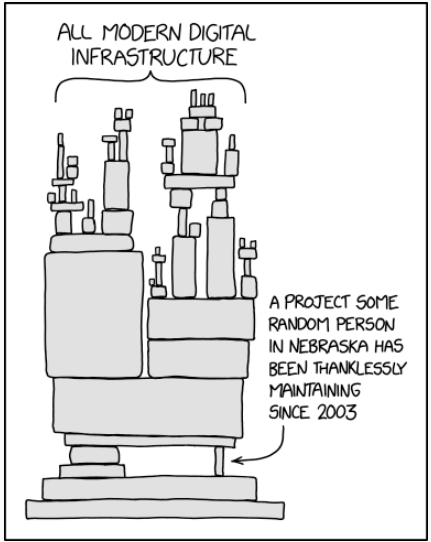
Open-Source Licenses

Licence	Author	Latest version	Publication date	Linking	Distribution	Modification	Patent grant	Private use	Sublicensing	TM grant
Academic Free License[11]	Lawrence E. Rosen	3.0	2002	Permissive	Permissive	Permissive	Yes	Yes	Permissive	No
Affero General Public License	Affero Inc	2.0	2007	Copylefted[12]	Copyleft except for the GNU AGPL[12]	Copyleft[12]	?	Yes[12]	?	?
Apache License	Apache Software Foundation	2.0	2004	Permissive[13]	Permissive[13]	Permissive[13]	Yes[13]	Yes[13]	Permissive[13]	No[13]
Apple Public Source License	Apple Computer	2.0	August 6, 2003	Permissive	?	Limited	?	?	?	?
Artistic License	Larry Wall	2.0	2000	With restrictions	With restrictions	With restrictions	No	Permissive	With restrictions	No
Beerware	Poul-Henning Kamp	42	1987	Permissive	Permissive	Permissive	No	Permissive	Permissive	No
BSD License	Regents of the University of California	3.0	?	Permissive[14]	Permissive[14]	Permissive[14]	Manually[14]	Yes[14]	Permissive[14]	Manually[14]
Boost Software License	?	1.0	August 17, 2003	Permissive	?	Permissive	?	?	?	?
Creative Commons	Creative	1.0	2008	Public	Public Domain	Public Domain	No	Public Domain	Public Domain	No

https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licences

But... Why Contributing?

Why Contributing?



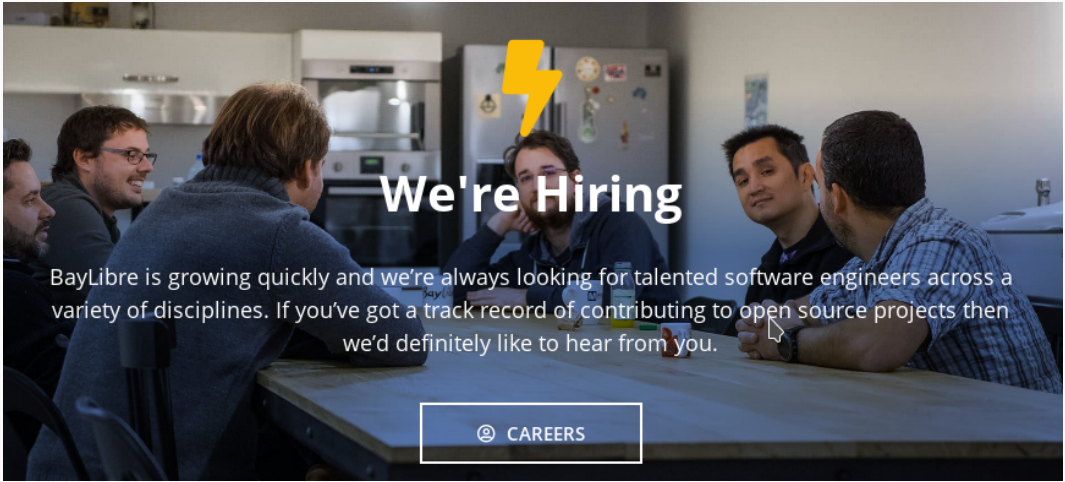
https://xkcd.com/2347/

Why Contributing?

Benefits for yourself:

- Improve your (transferable) skills:
  - Better coding, better workflow, learn new tricks
  - Practice code review
  - Jump into big projects
  - Collaboration
- Keep your knowledge up-to-date
  - Learn new languages and tools (Git)
- Get your dream job?
  - Grow your network
  - Nice line on your resume
  - Selling point in the interview

Open-Source Career?



https://baylibre.com/careers/

Why Contributing?

Benefits for your project (hobby or professional)

- Improve your code (peer review)
- Feedback from experts
- Lower maintenance cost
- Contribute back

Why NOT Contributing?

Why people do not contribute

- Intimidating
  - First time is hard - but don’t worry
- Keep your secret "sauce"... secret (lame!)
  - GPL-2 / GPL-3 require you to publish modifications anyway
- Boss do not want
  - Convince him!
- Time consuming process
  - True for the first contributions

I Need a Contribution

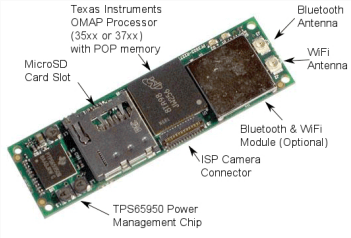
How to Get Started

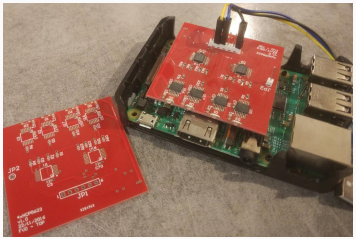
Many ways to do code

- Fix a bug
- Work on staging drivers (drivers/staging/)
- Add a new driver
- Add / improve board support
- Support older hardware (if you have it)
- Improve performances
- Reverse-engineer closed-source hardware (e.g. GPU)
- Look at mailing lists / bug trackers / etc. for ideas
    - Projects often have a "newcomers issues" list

How to Get Started

Some of my projects:





Not Just Code

Many contributions beyond code:

- Testing
  - Report a bug
  - Writing tests cases
  - Running tests (other hardware / etc.)
  - Debugging failing tests
- Review
- (Distro) packaging
- Documentation
- Translations
- Graphical elements / Art
- Bug triage
- Release manager
- Public advocate, fund raising, etc.

How _NOT_ to Get Started

Bad ways to get involved

- Engaging in counter-productive discussions (trolling)
- Being arrogant, aggressive, etc.
- Ignore comments

Toxic behavior is not longer tolerated (Linux Code of Conduct)

> *What the F*CK, guys?*
> *This piece-of-shit commit is marked for stable, but you clearly never even test-compiled it, did you?*

Linus Torvald (`https://lkml.org/lkml/2013/7/13/132`)

What Can Go Wrong?

Not always easy

- Maintainer can reject
- Maintainer is often overloaded / unresponsive
- Diverging views $\rightarrow$ infinite debate loop
- Motivation exhaustion
- Superseded by another change

Be prepared to fail, but do not give up!

Let's Do It

Contribution Blueprint

General process for most projects

1. Read the documentation, coding style, etc. (if any...)
2. Check issues / pull requests
3. (optional) Get in touch (ticket / mail / Discord / ...)
4. Checkout the latest source code
5. Repeat 1...N times:
    - Code ... compile ... commit ... test
    - Is it ready? Author, commit message, breakdown, unit tests...
    - Send changes
    - Address comments (be open-minded)
6. Celebrate

General Advices

Avoid these common mistakes:

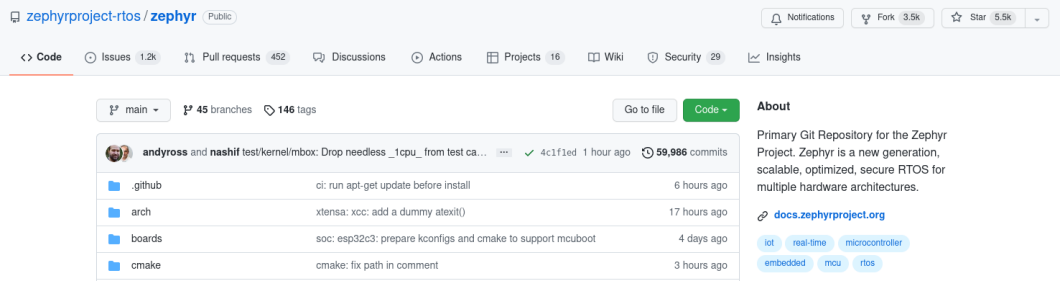
- Check ALL the documentation on the contribution process (coding style, etc.)
- Check for any prior work
  - Make sure that the change do not already exist or was rejected
- Look at past contributions for examples
- Use the latest (upstream) version
- Run unit tests if any

Let's Do It

Case 1: Github Projects

Github Projects

- World largest source code host
- 100+ million developers
- 420+ million repositories



https://en.wikipedia.org/wiki/GitHub

https://github.com/about

Github workflow

1. Fork the project
2. (optional) Create an issue to discuss the topic
3. Create a branch from the development branch
4. Code / commit / test
5. Push this branch to your GitHub project
6. Open a Pull Request on GitHub
7. Discuss and address comments, push the new version
8. The project maintainer merges the Pull Request

https://git-scm.com/book/en/v2/GitHub-Contributing-to-a-Project

Let's Do It

Case 2: Linux Kernel

Linux Kernel

Specifics of Linux kernel

- Huge project: 35 million lines in v5.18 (2022)
- Well-defined release cycles
- Many maintainers + thousands of contributors
  - No central forge !
- Email-based workflow (no joke)
  - Example

Linux RTFM (Read the Fine Manual)

The process is well documented:

- Kernel Development Process
- Coding style
- Howto
- Submitting patches
- Submission checklist

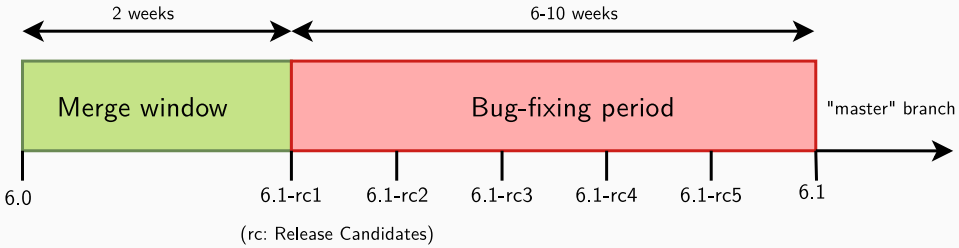
Plenty of other ressources:

- See at the end of slides

Linux Release

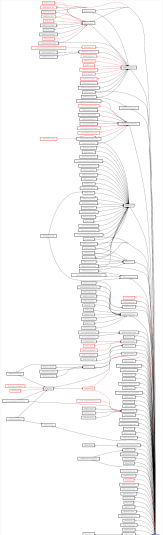
Typical schedule

- Merge window: 2 weeks
- -rc kernels: 6 - 10 weeks



Linux Maintainers

Tree of maintainers for v5.18:



https://lwn.net/Articles/895800/

Git Trees

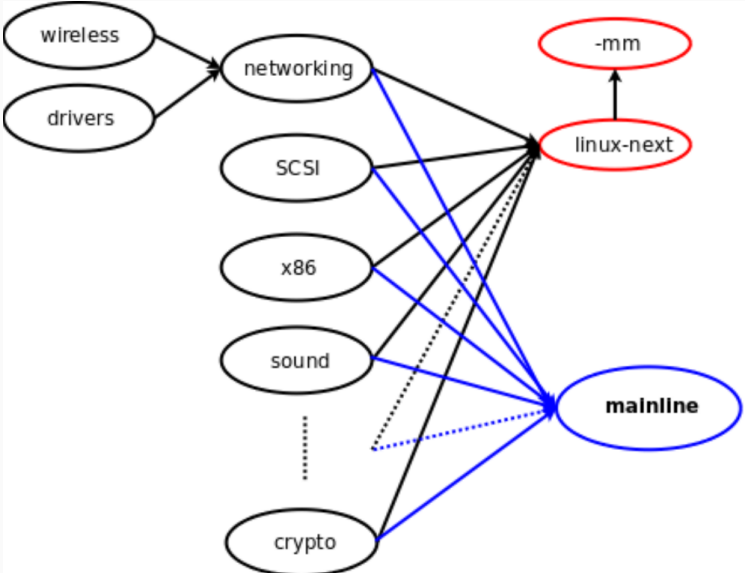
Which Git tree to choose ?

- Linus Torvald (official release)
- Maintainer
- linux-next

My advice:

- Simple change: official release
- Change in one subsystem: maintainer's tree
- More complex: linux-next
- Try to avoid vendor trees

Git Trees


wireless
drivers
networking
SCSI
x86
sound
crypto
-mm
linux-next
mainline

Get The (Correct) Source

Look-up MAINTAINERS

```
./scripts/get_maintainer.pl --scm -f path-to-file-or-directory
```

Example:

```
$ ./scripts/get_maintainer.pl --scm -f drivers/net/ethernet/ti/davinci_emac.c
Grygorii Strashko <grygorii.strashko@ti.com> (reviewer:TI ETHERNET SWITCH DRIVER (CPSW))
"David S. Miller" <davem@davemloft.net> (maintainer:NETWORKING DRIVERS,commit_signer:1/1=
Jakub Kicinski <kuba@kernel.org> (maintainer:NETWORKING DRIVERS,commit_signer:1/1=100%,au
linux-omap@vger.kernel.org (open list:TI ETHERNET SWITCH DRIVER (CPSW))
netdev@vger.kernel.org (open list:TI ETHERNET SWITCH DRIVER (CPSW))
linux-kernel@vger.kernel.org (open list)
git git://git.kernel.org/pub/scm/linux/kernel/git/netdev/net.git
git git://git.kernel.org/pub/scm/linux/kernel/git/netdev/net-next.git
git git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

Prepare Your Changes

Work in a branch

```
git remote add net-next git://git.kernel.org/pub/scm/linux/kernel/git/netdev/net-next.gi
git checkout -t -b my-net-changes-v1 net-next/master
```

Code … compile … commit … test (repeat)

```
git add <file1> ... <fileN>
git commit -s
```

- Describe your change (what / why / how)
- Pro tip: use `git log path/to/subsystem/` for some inspiration

Prepare Your Changes

Signed-off-by: Firstname Lastname <firstname.lastname@freedom.org>

- Developer's Certificate of Origin:
  Certify that you have the right to submit under the license
- Use your real name / e-mail for legal reasons
- Also: Acked-by, Reviewed-by, Reported-by, Tested-by
- See https://docs.kernel.org/process/submitting-patches.html

Patchset

- One feature per patchset
- One logical change per commit
- Each commit must apply / compile / work
- `git rebase -i <base\_commit>`
  (https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History)

Format Patches

Generate patches

For a single patch:

```
git format-patch --base=<base> <base>
```

For a patchset:

```
git format-patch -n --cover-letter --base=<base> <base>
```

Example:

```
$ git format-patch -n --cover-letter --base=auto HEAD~3
0000-cover-letter.patch
0001-iio-potentiometer-mcp4531-Add-support-for-MCP454x-MC.patch
0002-iio-potentiometer-mcp4531-Add-device-tree-binding-do.patch
0003-iio-potentiometer-mcp4531-Add-device-tree-binding.patch
```

Last Checks

Write cover letter (0000-cover-letter.patch)

- Not necessary when sending a single commit (omit `--cover-letter`)

Triple-check patches

- Commit message
- Typos
- `Signed-off-by` and other tags

Check coding style (tabs, 80 characters, braces,...)

```
./scripts/checkpatch.pl --strict *.patch
```

https://docs.kernel.org/process/coding-style.html

Send It !

Get a list of recipients (MAINTAINERS)

```
./scripts/get-maintainer.pl --git *.patch
```

Look up for maintainers, main contributors and mailing lists

Send plain text emails

```
git send-email --quiet --compose --no-signed-off-by-cc \
    --to "XZY <xzy@example.com>" --cc "linux-kernel@vger.kernel.org" \
    *.patch
```

https://docs.kernel.org/process/email-clients.html

- Test sending the patches to yourself first
- Email clients / providers can corrupt the format
  - Exchange servers are cursed

Review Process

- Done through email
  - Only plain text (***no*** HTML)
  - Do not top-post
- Be patient
  - Patches are rarely accepted in v1
- Be polite
  - Maintainers often have good reasons, try to understand "why"
  - They can be wrong: show facts
  - Always thanks reviewers
- Submit [PATCH v2]
- Wash, rinse, repeat... until accepted

Conclusion

Conclusion

To be successful

- Start small
- RTFM before submitting
- Be humble
- Be prepared to learn a lot

Resources

Resources

Linux kernel newbies

- https://kernelnewbies.org/FirstKernelPatch
- https://kernelnewbies.org/PatchPhilosophy

Presentations from Linux conference

- *From an Idea to a Patch in the Linux Mainline* (2021)
  - Slides
  - Youtube
- *How not to submit a patchset?* (2023)
  - Kernel Recipes

Resources

Mailing lists:

- Existing lists and subscribe: http://vger.kernel.org/vger-lists.html
- Archives: https://lore.kernel.org/lists.html
  - LKML: https://lore.kernel.org/lkml/
  - linux-arm: https://lore.kernel.org/linux-arm-kernel/