Linaro

connect
San Francisco 2015

SFO15-T2: Upstreaming 101

Presented by

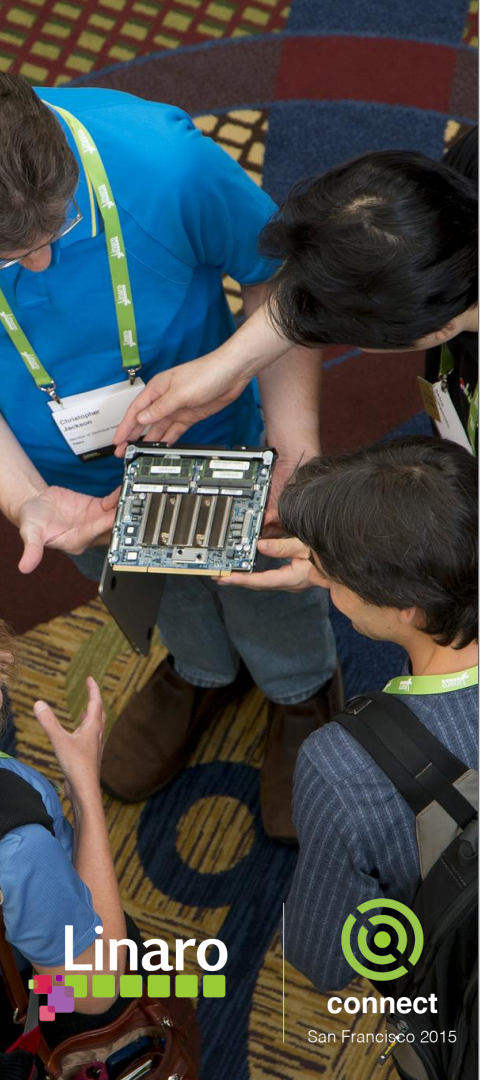
Daniel Thompson

Date

Tuesday 22 September 2015

Event

SFO15



Overview

- Focus is on **Linux kernel** upstreaming
- What is upstreaming?
  - Define what it is first
- How to upstream?
  - Process and mechanics
- Target audience
  - Developers
  - Engineering managers

Prerequisites

- Familiar with source code control concepts
- Familiar with git terminology (pulls, topic branches, etc.)
- Technical understanding of kernel level software

What is upstreaming?

- Linux kernel context
- Upstream means to move software into the top level Linux repository
- This is Linus Torvalds' Linux repository (aka “mainline”)

What is mainline?

The Linux Kernel Archives

About Contact us FAQ Releases Signatures Site news

Protocol	Location
HTTP	https://www.kernel.org/pub/
FTP	ftp://ftp.kernel.org/pub/
RSYNC	rsync://rsync.kernel.org/pub/

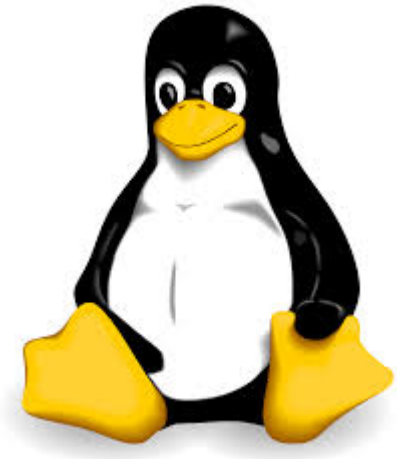
Latest Stable Kernel: 3.16.2

mainline:	3.17-rc4	2014-09-07	[tar.xz]	[pgp]	[patch]	[view patch]		[cgit]
stable:	3.16.2	2014-09-05	[tar.xz]	[pgp]	[patch]	[view patch]	[view inc]	[cgit] [changelog]
stable:	3.15.10 [EOL]	2014-08-14	[tar.xz]	[pgp]	[patch]	[view patch]	[view inc]	[cgit] [changelog]
longterm:	3.14.18	2014-09-05	[tar.xz]	[pgp]	[patch]	[view patch]	[view inc]	[cgit] [changelog]
longterm:	3.12.28	2014-09-07	[tar.xz]	[pgp]	[patch]	[view patch]	[view inc]	[cgit] [changelog]
longterm:	3.10.54	2014-09-05	[tar.xz]	[pgp]	[patch]	[view patch]	[view inc]	[cgit] [changelog]
longterm:	3.4.103	2014-08-14	[tar.xz]	[pgp]	[patch]	[view patch]	[view inc]	[cgit] [changelog]
longterm:	3.2.62	2014-08-06	[tar.xz]	[pgp]	[patch]	[view patch]	[view inc]	[cgit] [changelog]
longterm:	2.6.32.63	2014-06-18	[tar.xz]	[pgp]	[patch]	[view patch]	[view inc]	[cgit] [changelog]
linux-next:	next-20140912	2014-09-12						[cgit]

https://kernel.org

Who Exactly Contributes to Mainline?

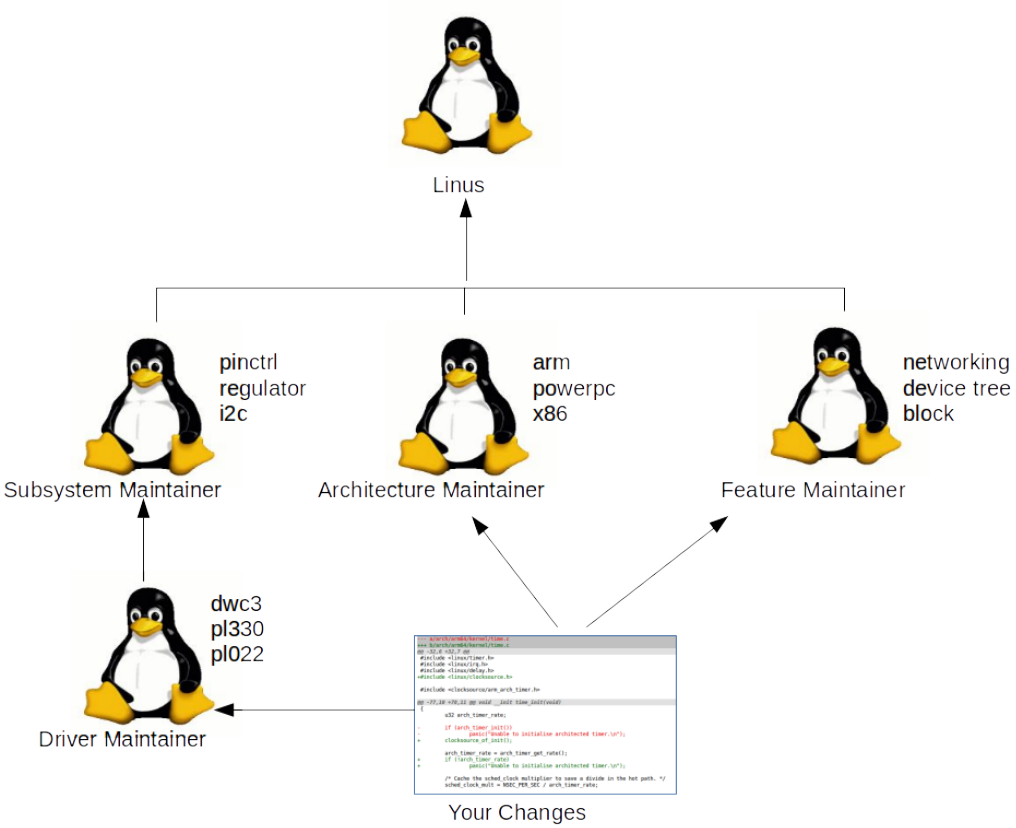
from list of top 4.2 contributors: http://www.remword.com/kps_result/4.2_whole.html



Swimming upstream to mainline

- Distinct hierarchy of repositories
- Repositories are git trees
  - One or more topic branches that feed into the mainline kernel
- Different owners for each repository in the tree

Upstream code flow



Maintainers

- Component code owners
  - Subsystem
  - Driver(s)
  - Filesystem
  - Architecture/platform
- Responsible for a slice of the kernel tree
- Gatekeepers
  - Control acceptance of incoming patches
  - Acceptance criteria varies

Maintainer numbers

- 1033 unique maintainers in v4.2

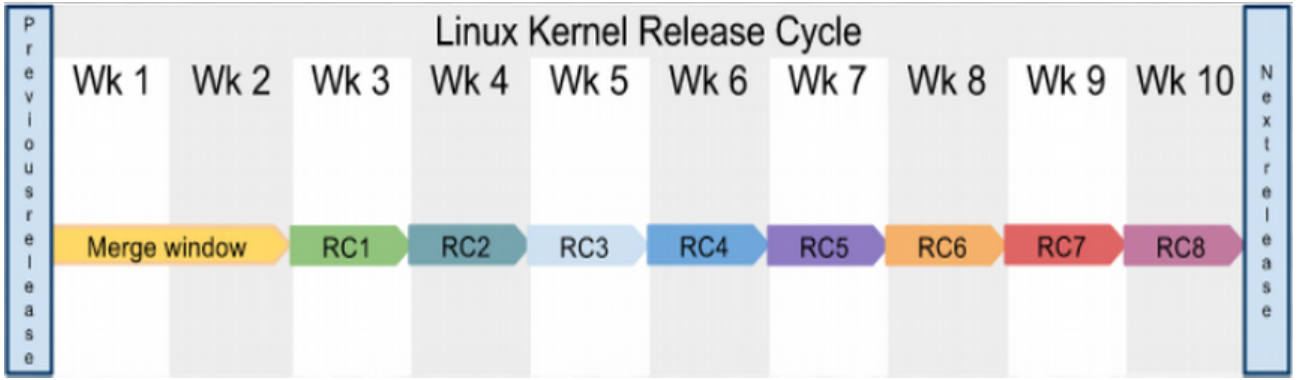
```
$ grep "^M:.*" MAINTAINERS | sort | uniq | wc -l
1033
```

- Each subsystem/component has one or more maintainers
- Example MAINTAINERS entry:

```
ARM PORT
M:      Russell King <linux@arm.linux.org.uk>
L:      linux-arm-kernel@lists.infradead.org …
W:      http://www.arm.linux.org.uk/
S:      Maintained
F:      arch/arm/
```

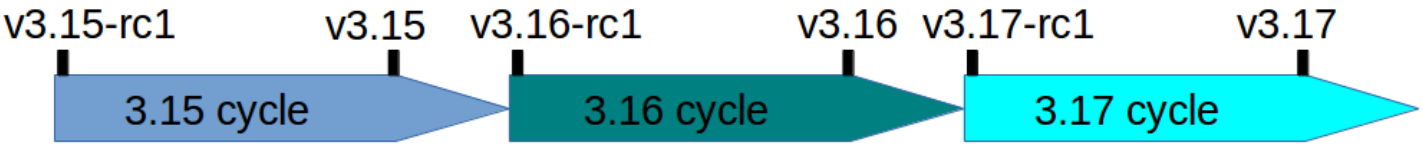
Understanding Merge Windows

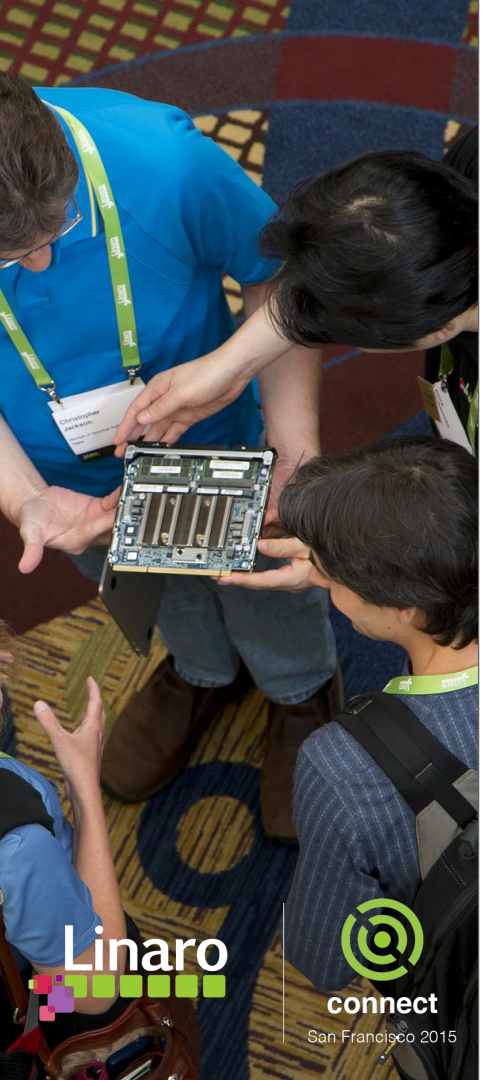
- Merge windows open every 10 weeks +/- 1 week
- Merge window is open for 2 weeks
- New functionality is only taken into Linus Torvalds' tree during the merge window



Understanding Merge Windows

- Merge window planning
  - New functionality needs to be accepted in maintainer trees usually by the -rc6 or -rc7 release
  - After -rc7 most maintainers will only be accepting fixes
- Less than 7 weeks after a merge window closes to have a maintainer queue a patch for the next merge window.





How to Upstream?

- Preparation
- Creation
- Posting
- Feedback
- Maintenance
- How Long Does it Take?

Preparation

- Know your content
  - Your contribution fits into a kernel framework. What is it?
  - Write your contribution to conform to the current framework standards and kernel APIs
- Know who else is doing work in your area upstream
  - Is anybody doing work related to the framework that could affect framework APIs?

Preparation

- Review *Documentation/** for clarification on APIs and frameworks
- Review *Documentation/devicetree/bindings/** for clarification on Device Tree bindings and best examples
- Read devicetree mailing list to learn about DT best practices
  - http://vger.kernel.org/vger-lists.html#devicetree

Preparation

- On what mailing lists and IRC channels are similar contributions discussed?
  - Follow these forums and understand the direction the frameworks are moving in APIs and style.
  - Ask questions, if necessary, to clarify what APIs to make use of before writing your code.
- Read *linux-arm-kernel*, at a minimum
  - http://lists.infradead.org/mailman/listinfo/linux-arm-kernel
- *#armlinux* on freenode for ARM kernel discussions

Preparation

- Read and understand
  - *Documentation/SubmittingPatches*
  - *Documentation/SubmitChecklist*
  - *Documentation/devicetree/bindings/ABI.txt*
  - *.../devicetree/bindings/submitting-patches.txt*
  - *Greg Kroah-Hartman, "How to piss off a kernel subsystem maintainer".*
    http://www.kroah.com/log/linux/maintainer.html
    http://www.kroah.com/log/linux/maintainer-02.html
    http://www.kroah.com/log/linux/maintainer-03.html
    http://www.kroah.com/log/linux/maintainer-04.html
    http://www.kroah.com/log/linux/maintainer-05.html
    http://www.kroah.com/log/linux/maintainer-06.html

Creation

- Use git for code management
- Logical division of commits
  - Small changes
  - Functionality
  - Individually complete (bisectability)
- Logical commits allow for ease of review and speed acceptance

Creation

- Multipart series subject line
  - *Subject: [PATCH 01/11] subsystem: summary phrase*
- Version 3 of a single patch submission
  - *Subject: [PATCH v3] subsystem: summary phrase*
- RFC patch submission
  - *Subject: [PATCH RFC] subsystem: summary phrase*

Creation

- Take time to create a quality commit log message
  - Why the patch is needed
  - What the patch implements
  - How the patch is implemented.
  - *“The conditional in foo() did not handle case bar and broke platform baz. Add an additional conditional and error path to foo() to handle bar.”*
- Each commit **must** have a well-formed commit log

Creation

- Create patches with *git format-patch*
  - *--cover-letter* for a patch series
  - The cover letter contains an overview describing the purpose and scope of the entire series.
- Use *scripts/checkpatch.pl* to verify coding style and semantics
- Use *scripts/get_maintainer.pl* to verify maintainer list for submission.

Posting

- Post patch or patch series
  - Maintainers on To:
  - Mailing lists on Cc:
  - Other interested parties on Cc:
- Use *git send-email* to post patches/series
- Expect comments!

Feedback on Mailing Lists

- No response
  - Be patient, maintainers are very busy
  - Wait one week to resend if no response
- Tough questions
  - Be prepared to justify your decisions or approach in great detail
  - Maintainers aren't always correct, be strong and concise in your justifications
  - If you don't understand a comment, ask for clarification
  - Don’t ignore comments!

Feedback on Mailing Lists

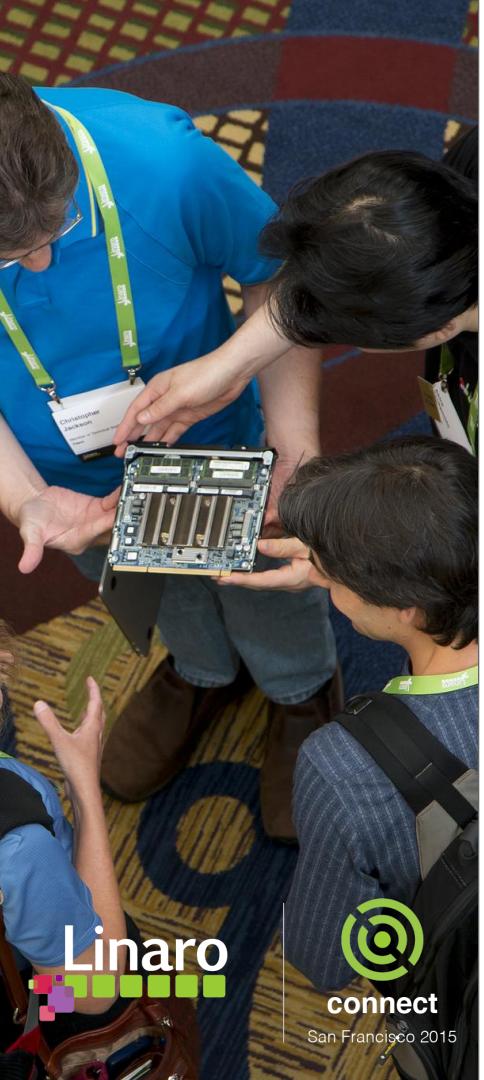
- Use a sane email client
  - Plain text wrapped at 72 columns (unless its a diff)
  - Working threading
  - Saves messages in a format git understands
  - Advice on configuring various mail user agents
    - *Documentation/email-clients.txt*
- Getting flamed
  - No need to worry about this if you are following the documented practices.

Feedback on Mailing Lists

- Making changes
  - Be responsive! Address comments via discussion and come to a conclusion quickly
  - Incorporate agreed upon comments and quickly submit a new version
  - Be prepared to not get an acceptable comment resolution on the first try
  - Expect **many** iterations
- Resubmission
  - Increment the version number in the subject line for the patch series and include a changelog

Maintenance

- Once accepted, now what?
  - Need to follow mailing lists for upcoming changes
  - Help review any new changes within the same area as your contribution
  - Test, test, test



Summary

- Preparation is key to success
- RTFM on everything
- Ask questions
- Act with a sense of urgency on comments
- Understand merge window timing