Joining the herd of cats: how to work with the kernel development process

Jonathan Corbet
corbet@lwn.net

Slides: http://lwn.net/talks/elc2007/

I:
Introduction

Why?

“There are a number of very good Linux kernel developers, but they tend to get outshouted by a large crowd of arrogant fools. Trying to communicate user requirements to these people is a waste of time. They are much too 'intelligent' to listen to lesser mortals.”

-- Jack O'Quin, Linux audio developer

Why?

There is great value to working with the community

- Influence development directions
- Offload code maintenance
- Better support for customers
- More efficient development
- Benefit from community expertise
- Take ownership of your platform – and make it better

Free software is different

Proprietary software	Free software
Product driven	Process driven

Free software is different

Proprietary software	Free software
Product driven	Process driven
Top-down requirements	Bottom-up requirements

Free software is different

Proprietary software	Free software
Product driven	Process driven
Top-down requirements	Bottom-up requirements
Short time horizon	Long-term view

Free software is different

Proprietary software	Free software
Product driven	Process driven
Top-down requirements	Bottom-up requirements
Short time horizon	Long-term view
Internal QA	External QA

Free software is different

Proprietary software	Free software
Product driven	Process driven
Top-down requirements	Bottom-up requirements
Short time horizon	Long-term view
Internal QA	External QA
Hierarchical decisions	Consensus decisions

Free software is different

Proprietary software	Free software
Product driven	Process driven
Top-down requirements	Bottom-up requirements
Short time horizon	Long-term view
Internal QA	External QA
Hierarchical decisions	Consensus decisions
Private	Public

Free software is different

Proprietary software	Free software
Product driven	Process driven
Top-down requirements	Bottom-up requirements
Short time horizon	Long-term view
Internal QA	External QA
Hierarchical decisions	Consensus decisions
Private	Public
Complete control	Little control

The kernel is even more different

It's big

Over 2,000 contributors 3/2006 to 4/2007

Only 10 contributed over 1% of changes

It's worldwide

Significant contributions from

- North America
- Europe
- Japan
- South America
- India
- ...

It's of great commercial interest

At least 2/3 of kernel work is by paid developers

It's growing quickly

750,000 lines added 3/2006 to 3/2007

It's the wild west

It's the wild west

...but that is changing

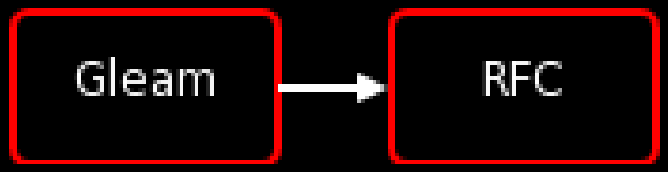
II:
Process
issues

Do: understand the patch lifecycle

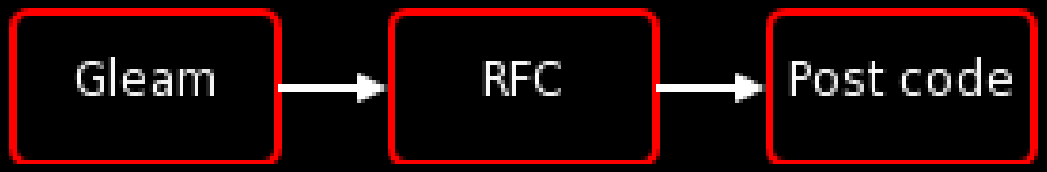
Much developer pain results from a failure to understand how code gets into the kernel.

Patch lifecycle: the beginning

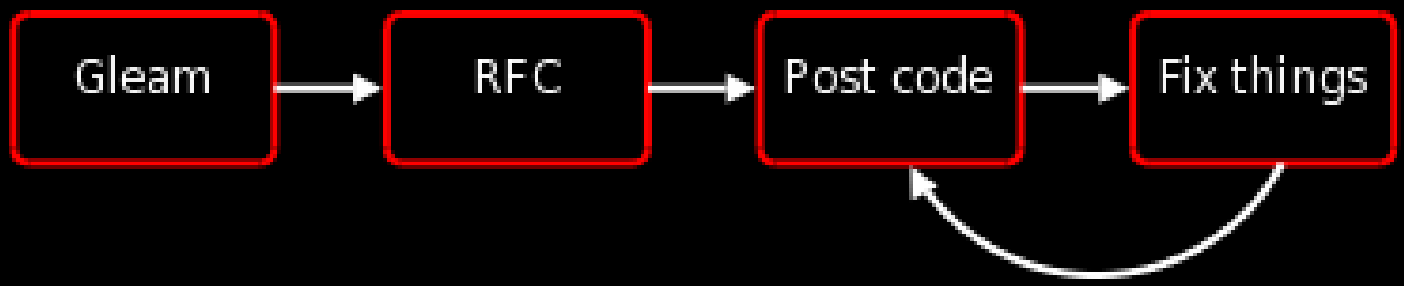
Patch Lifecycle: the RFC



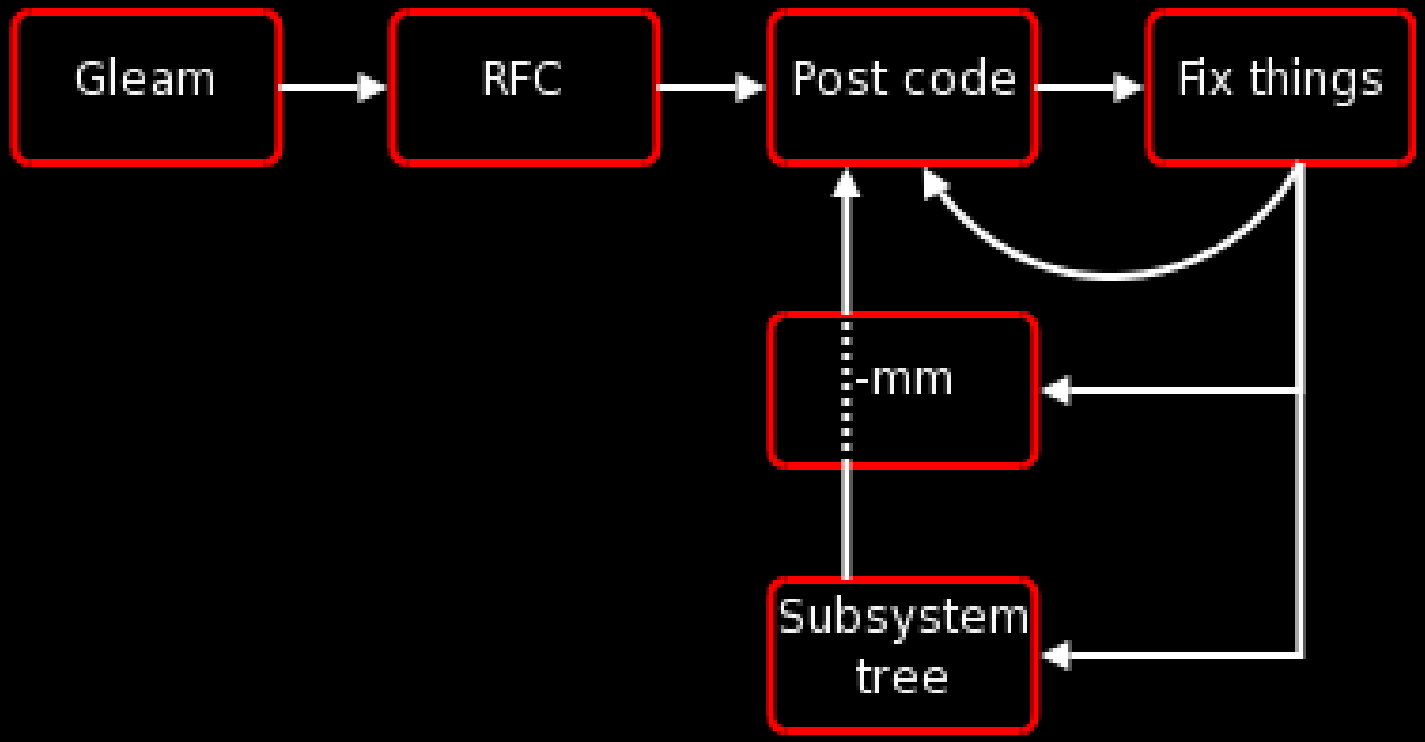
Patch lifecycle: first code



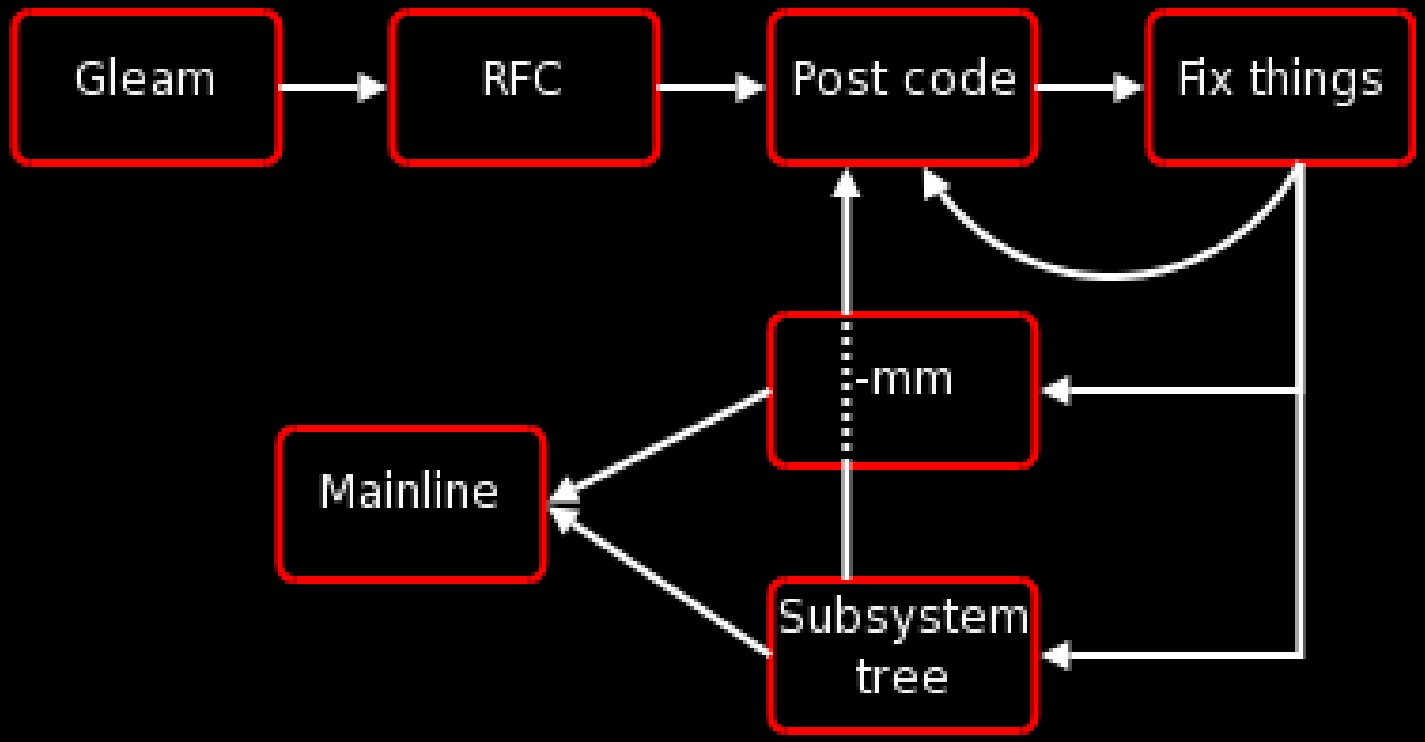
Patch lifecycle: repairs



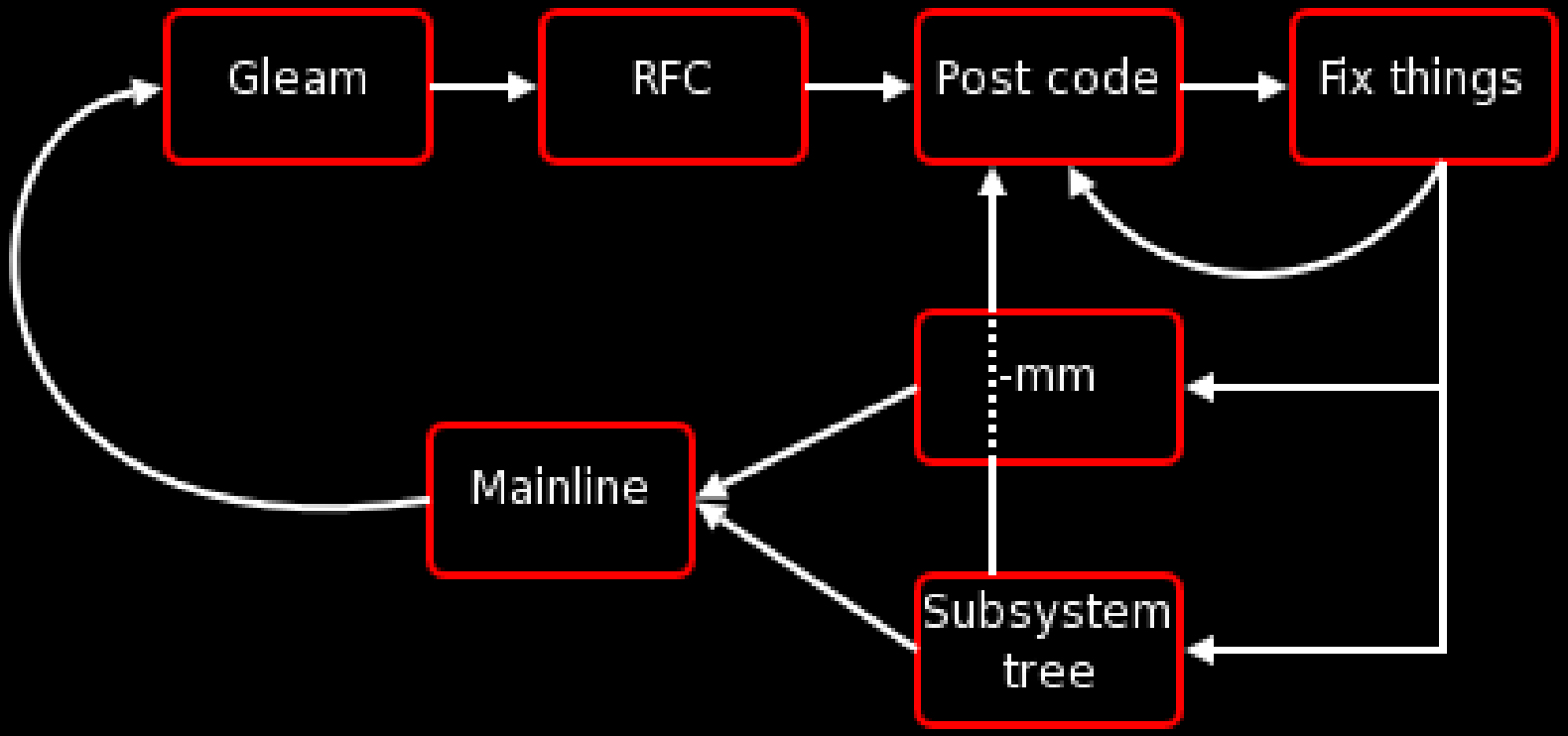
Patch lifecycle: wider testing



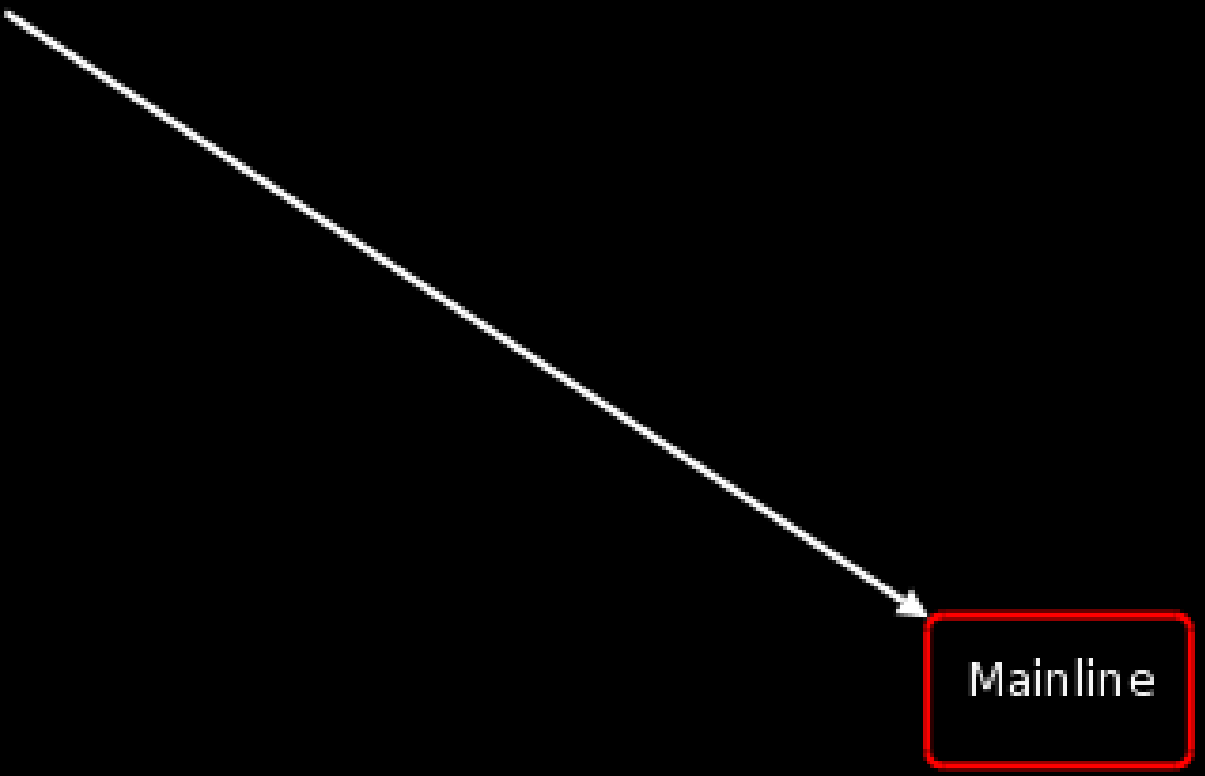
Finally: into the mainline



Patch lifecycle: repeat



Lifecycle: the corporate view



Do: communicate early

Let the community know what you are doing

Avoid duplication

Avoid mistakes

Do: release early

Big vendor mistake:

> “We'll release the code after it passes internal QA”

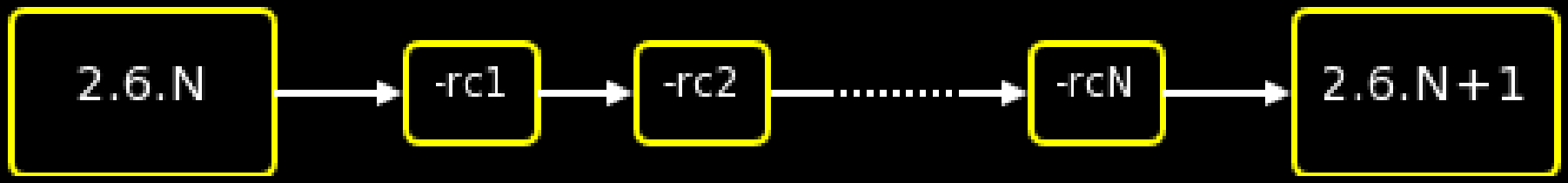
By then it is too late

Do: expect to make changes

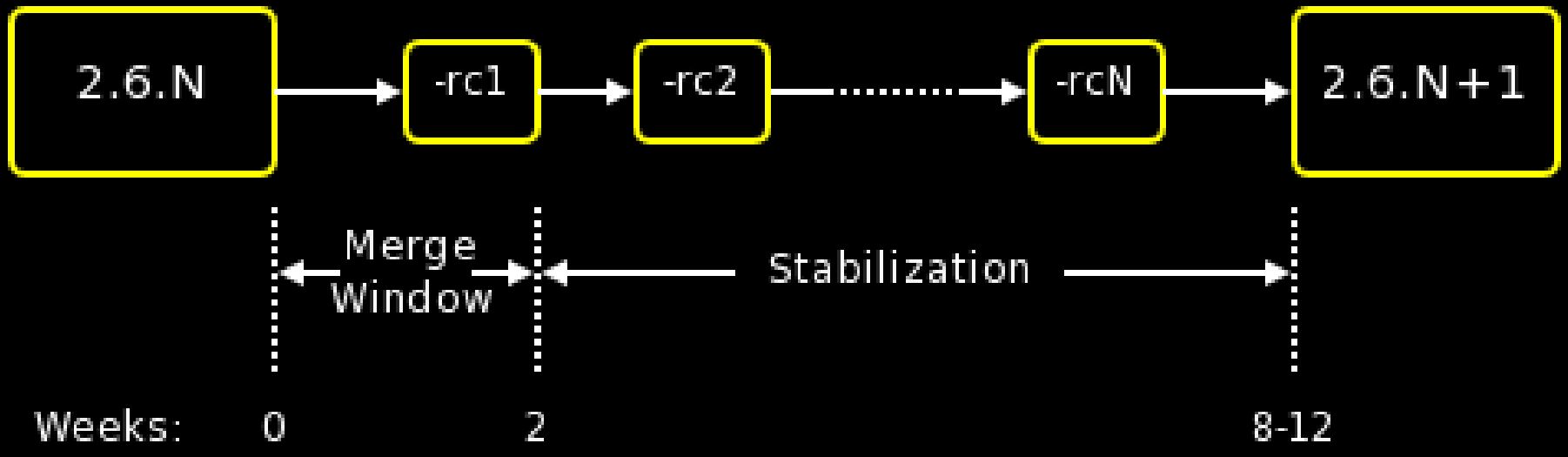
No initial code submission is perfect

Kernel developers have different goals

The kernel release cycle



The kernel release cycle



Do: observe the merge window

“I'm really fed up with having to pull big changes after the merge window, because it just doesn't seem to let up. I'm going to go postal on the next maintainer who doesn't understand what 'merge window' and 'fixes only' means”

-- Linus Torvalds

II:
Patch submission

Do: Send in your changes

Avoid having to carry changes out-of-tree

Draw attention to problems

Influence the direction of the kernel

Don't: send multipurpose patches

Patches should:

- Be small (if possible)
- Do exactly one thing

If you have a big change:

- Split it into independent pieces

Do: send bisectable patches

“git bisect” is a great tool for finding regressions

- Binary search on the patch stream

To support bisect:

- Your patch series must work after every patch

Do: take care in submitting patches

- Use diff -u
- No MIME attachments
- Describe them properly
  - A one-line summary at top
  - Longer description below (if needed)
  - Justify the patch
- Include a Signed-off-by: line
- Avoid word-wrapping
  - Thunderbird is especially bad here

See:

- Documentation/SubmittingPatches

Find the correct mailing list

linux-kernel is not the right place for all patches

Example: networking patches go to netdev

See:

- MAINTAINERS
- vger.kernel.org/vger-lists.html

Aside: the new linux-kernel subscriber

1) I want to be part of this, I'll subscribe

Aside: the new linux-kernel subscriber

1) I want to be part of this, I'll subscribe
2) I have mail from Linus! Cool!

Aside: the new linux-kernel subscriber

1) I want to be part of this, I'll subscribe
2) I have mail from Linus! Cool!
   ...I wish I understood it

Aside: the new linux-kernel subscriber

1) I want to be part of this, I'll subscribe
2) I have mail from Linus! Cool!
   ...I wish I understood it
3) Hmm...I have a *lot* of mail from Linus

Aside: the new linux-kernel subscriber

1) I want to be part of this, I'll subscribe
2) I have mail from Linus! Cool!
   ...I wish I understood it
3) Hmm...I have a *lot* of mail from Linus
   ...I thought he was a nicer guy than that.

Aside: the new linux-kernel subscriber

1) I want to be part of this, I'll subscribe
2) I have mail from Linus! Cool!
   ...I wish I understood it
3) Hmm...I have a *lot* of mail from Linus
   ...I thought he was a nicer guy than that.
4) 350 new messages. Ouch!
   ...I was only gone for an hour...

Aside: the new linux-kernel subscriber

1) I want to be part of this, I'll subscribe
2) I have mail from Linus! Cool!
   ...I wish I understood it
3) Hmm...I have a *lot* of mail from Linus
   ...I thought he was a nicer guy than that.
4) 350 new messages. Ouch!
   ...I was only gone for an hour...
5) I'll post my great idea for using C++ in the kernel

Aside: the new linux-kernel subscriber

1) I want to be part of this, I'll subscribe
2) I have mail from Linus! Cool!
   ...I wish I understood it
3) Hmm...I have a *lot* of mail from Linus
   ...I thought he was a nicer guy than that.
4) 350 new messages. Ouch!
   ...I was only gone for an hour...
5) I'll post my great idea for using C++ in the kernel
6) These people have no clue of how to program!

Aside: the new linux-kernel subscriber

1) I want to be part of this, I'll subscribe
2) I have mail from Linus! Cool!
   ...I wish I understood it
3) Hmm...I have a *lot* of mail from Linus
   ...I thought he was a nicer guy than that.
4) 350 new messages. Ouch!
   ...I was only gone for an hour...
5) I'll post my great idea for using C++ in the kernel
6) These people have no clue of how to program!
7) My full mailbox is making me miss those great penny stock alerts.

Aside: the new linux-kernel subscriber

1) I want to be part of this, I'll subscribe
2) I have mail from Linus! Cool!
   ...I wish I understood it
3) Hmm...I have a *lot* of mail from Linus
   ...I thought he was a nicer guy than that.
4) 350 new messages. Ouch!
   ...I was only gone for an hour...
5) I'll post my great idea for using C++ in the kernel
6) These people have no clue of how to program!
7) My full mailbox is making me miss those great penny stock alerts.
8) Unsubscribe. I'll go run Windows now.

On mailing lists

Ask whether you really need to read linux-kernel
- Many of us do

Consider a subsystem list instead

If so, read it sparingly
- Look for interesting topics and contributors

Do: listen to reviewers

Reviewing patches is hard, thankless work

When a reviewer makes a comment

- Say “thanks”
- Respond politely
- Fix the problem (or justify the current code)

Don't: attack reviewers

...even if they are rude

Don't: take criticism personally

Patch reviewers do not hate you

They do not hate your company

They do not hate your employees

Requests for major changes

Reviewers may ask for big changes
- Push functionality into higher layers
- Reimplement major functionality
- Clean up a longstanding mess

Their goals are different than yours
- Long-term maintainability is key

Try to accommodate these requests
- They usually make sense in the long term

Do: Let go

Others will patch your code

- An Acked-by: for good changes is polite

They may replace it outright!

- Consider it the sincerest form of flattery

Once you release code under a free license

- ...you no longer have control
- It gets better without work from you!

III:
Coding
issues

Do: follow the coding style

Documentation/CodingStyle

Do: avoid unnecessary abstractions

Things to avoid:

- HAL layers
- Unused parameters “just in case”
- Single-line functions

API stability

There is no stable internal kernel API
- Get used to it

Ways to cope
- Get your code into the mainline
- http://lwn.net/Articles/2.6-kernel-api

Don't: add multi-version code

Support the current mainline kernel

...and no others

Do: clean up your messes

Breaking an internal API is OK

...if there is a good reason for it

But:

you have the responsibility to fix in-tree code

Don't add regressions

...even to fix something else

Don't: change the user-space API

Breaking applications is bad news

The API includes

- System call behavior
- /proc files
- Sysfs files
- Netlink

Don't: assume all the world is a PC

Linux runs on all kinds of systems

- handhelds to supercomputers
- 32/64 bit
- Single processor through thousands of processors
- A few dozen architectures

Your code should build and work everywhere

In particular:

- Get your locking right from the beginning

Do: avoid silly mistakes

Use the tools:

- gcc -W
- lockdep
- fault injection framework
- slab poisoning
- sparse
- ...

Red flags

- #ifdef
- typedefs
- ioctl()
- Silence
- inline functions
- Heavy stack usage
- Unnecessary abstractions
  - HAL layers
  - Single-line functions
  - Unused parameters

IV:
Final
notes

Don't: submit tainted code

Read the Developer's Certification of Origin
- Be sure you mean it

Be very careful with reverse engineering
- Chinese wall approach should be used

Don't: ship binary-only modules

Legality of these modules is dubious

The benefit is even more dubious

Respect your customers: give them the source

Do: use the resources available

There is information and help out there

- kernelnewbies.org
- Kernel mentors
- Documentation/HOWTO
- LWN

Do: join in and have fun

Questions....?