How to contribute to the Linux kernel and not die trying

Javier Martinez Canillas
javierm@redhat.com
@martinezjavier

$ whoami

- Software Engineer @ Red Hat
- Linux and Fedora developer
- Linux contributor since 2010
- 1000+ contributions to the Linux kernel

Agenda

- Motivation
- Linux development process
- Contribution steps
  - Pitfalls
  - Good practices
  - Tools

Motivation

Motivation

Linux is the largest collaborative software project in the world.

Motivation

Due to the scale of the community, each maintainer has their own optimized workflow.

Motivation

It's a very costly operation for maintainers to diverge from their workflow.

Motivation

So even when there is a single community and documented development process...

Motivation

...there isn't a single way to submit a patch.

Motivation

There are different ways to submit patches to different subsystems.

Motivation

This talk shares my methods for minimizing this overhead for maintainers.

Linux Development Process

Linux development process

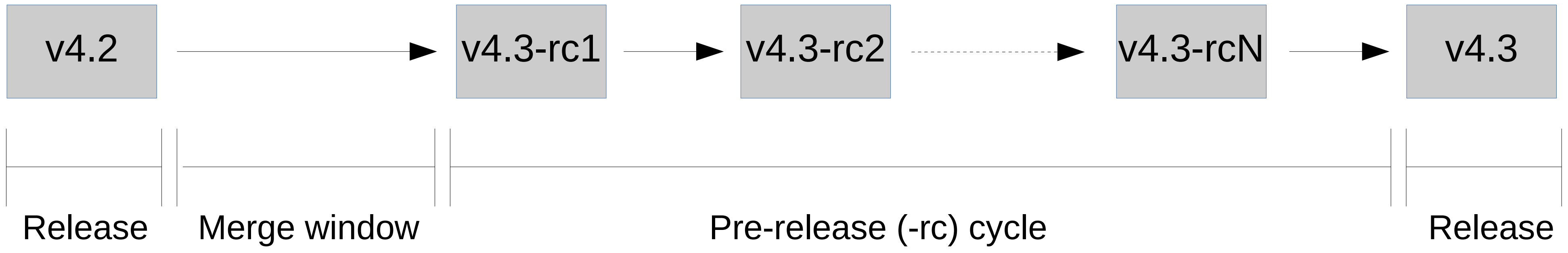
- Most projects use a feature based release model
- Linux instead uses a time based release model

Linux development process

“Linux is evolution, not intelligent design”

- Linus Torvalds

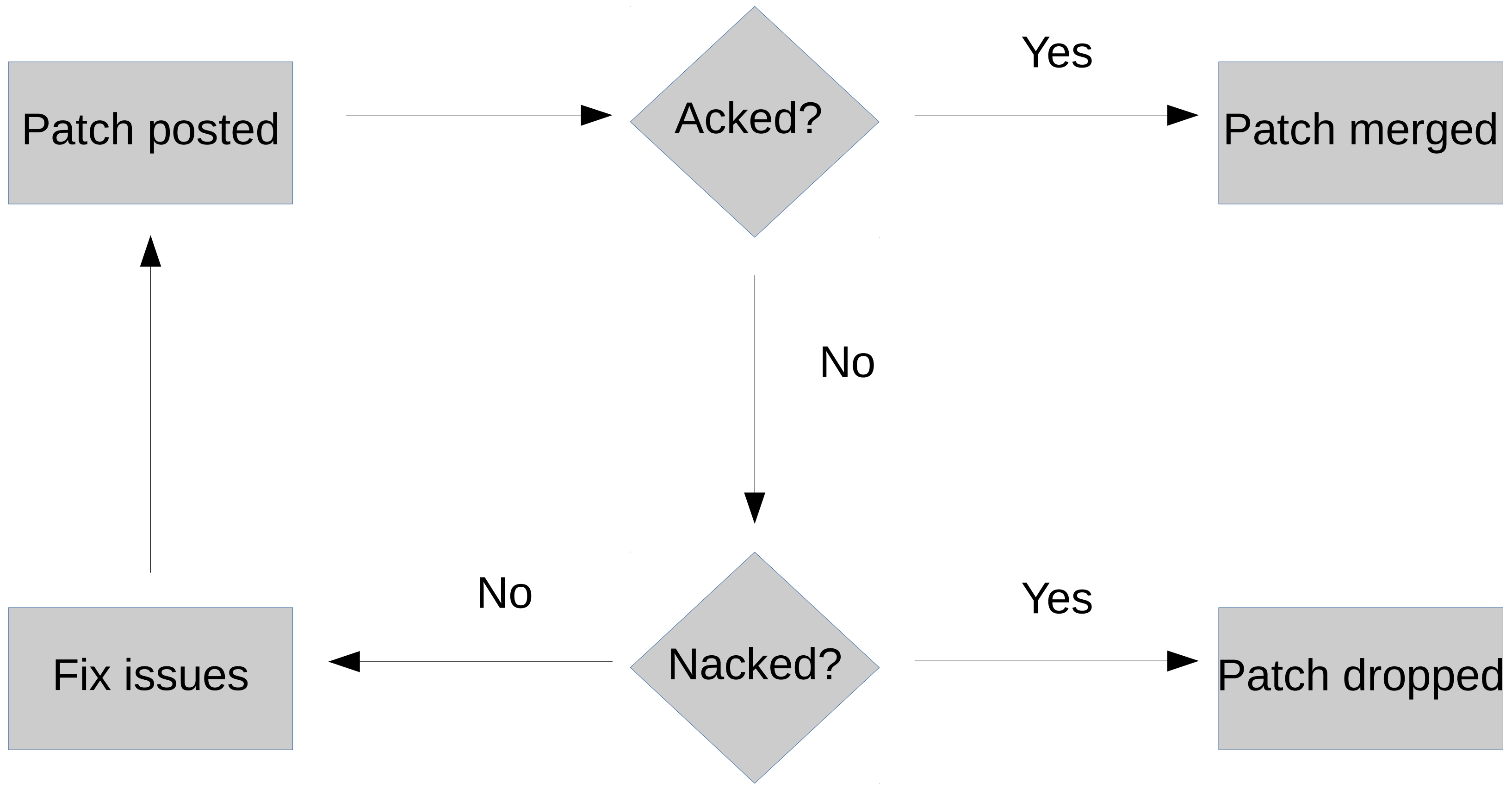
Linux kernel release cycle



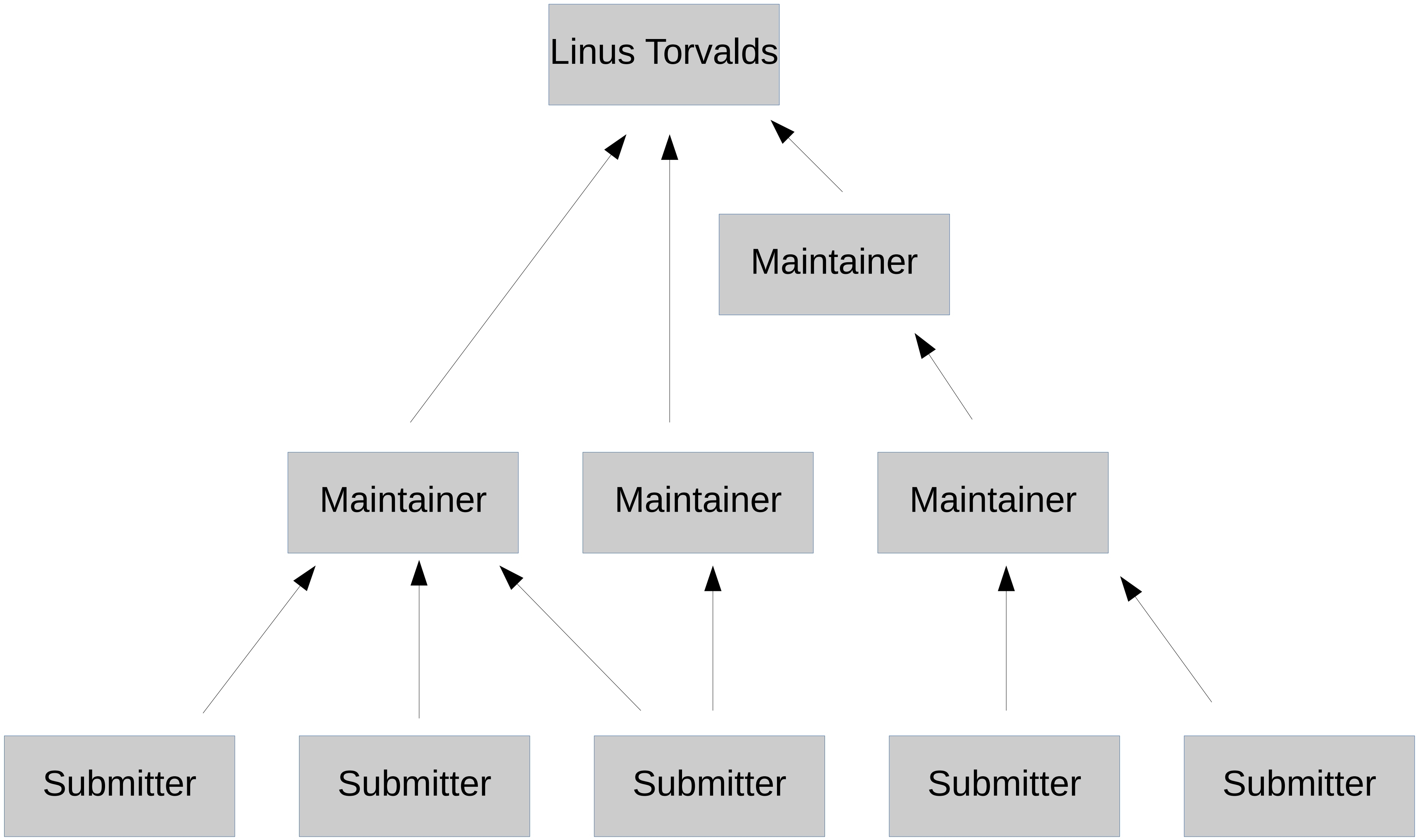
Linux kernel trees

- linux.git: Linus Torvalds' tree
  - git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
- linux-stable.git: contains previous versions on which fixes are backported
  - git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
- subsystem trees: each maintainer has a tree used for development
- linux-next.git: integrates all the subsystem maintainer trees for testing
  - git://git.kernel.org/pub/scm/linux/kernel/git/next/linux-next.git

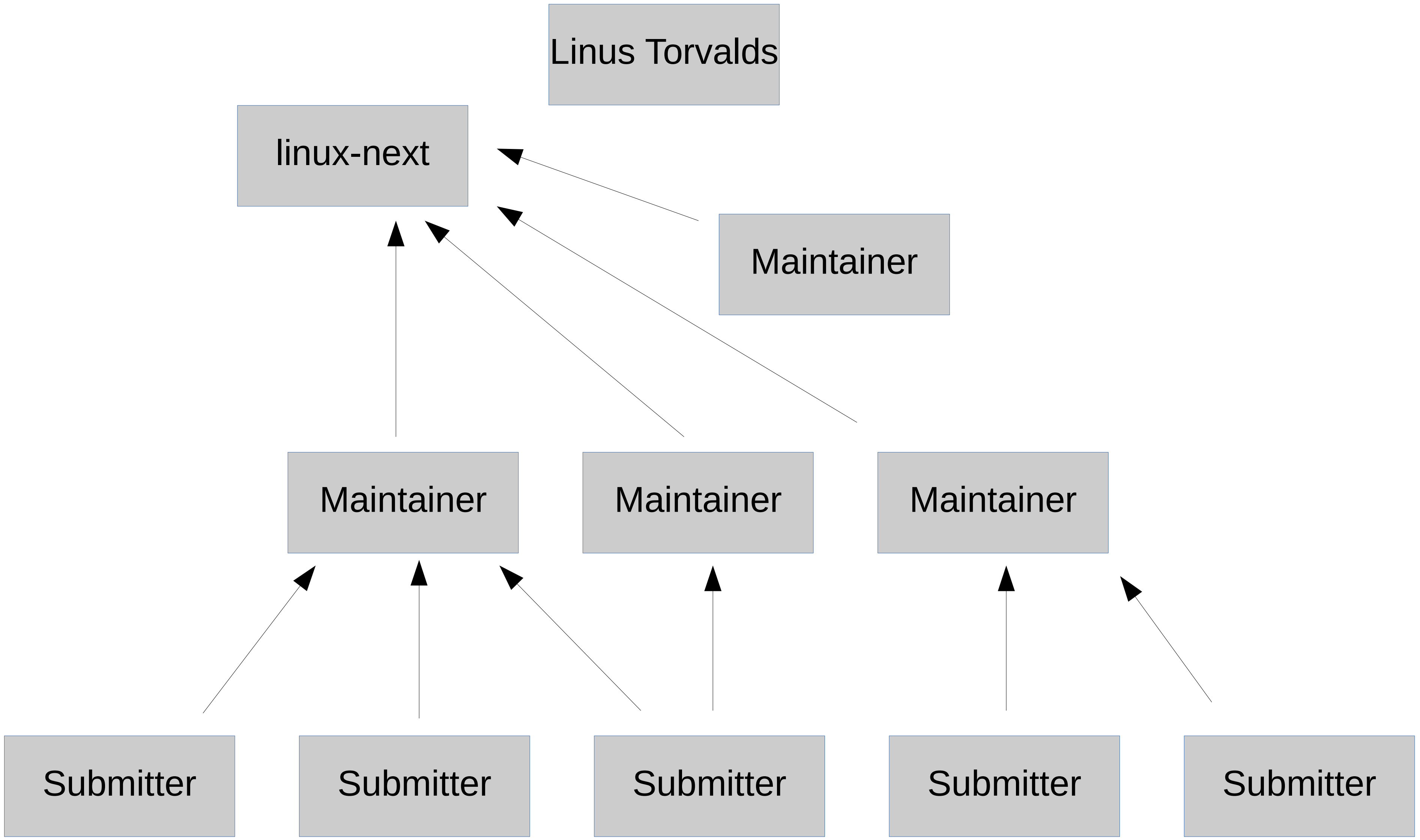
A patch flow to mainline



A patch flow to mainline

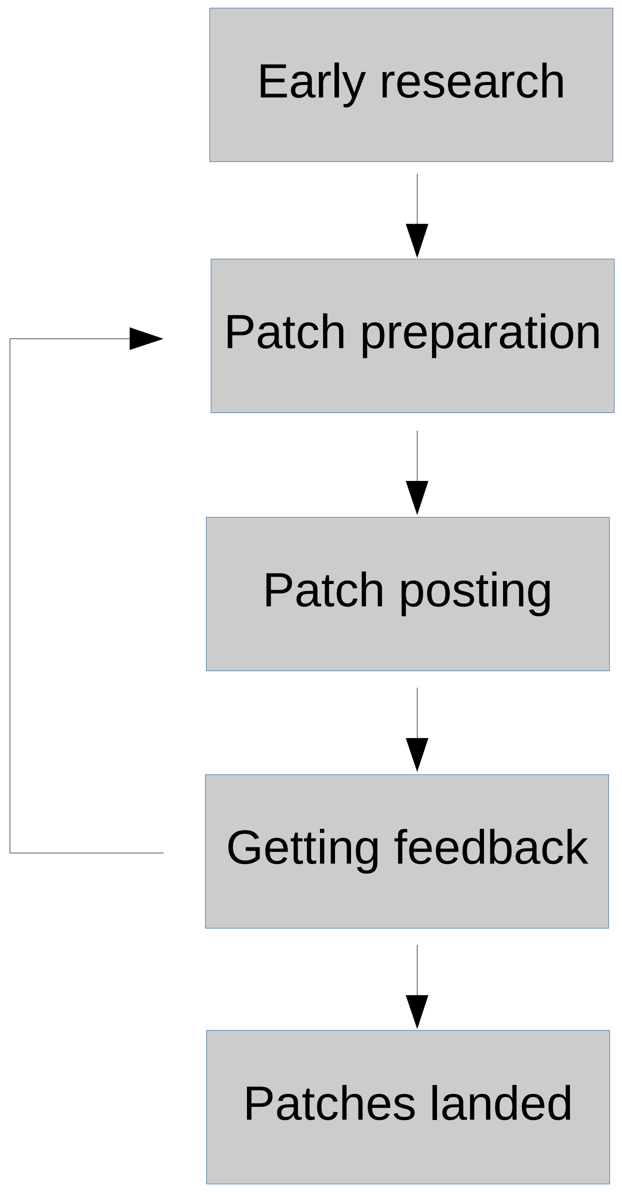


A patch flow to mainline

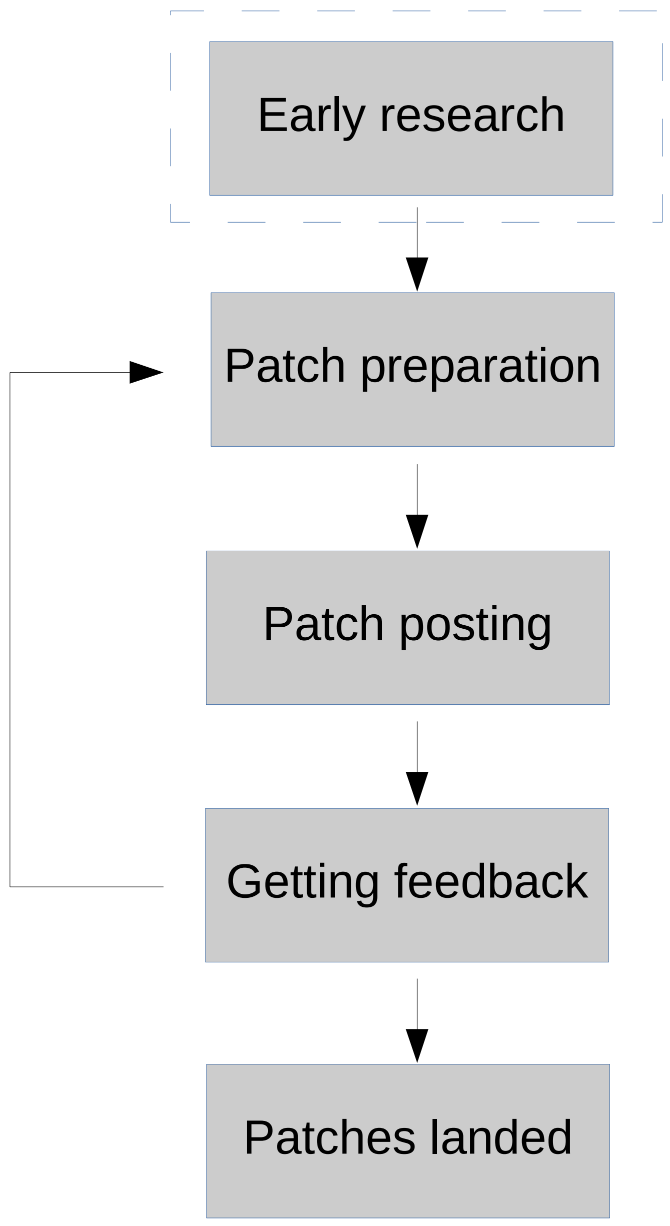


Contribution Steps

Contribution Steps



Contribution Steps



Early Research

- The development process must be understood before preparing a patch.
- This is one of the most important steps for a successful contribution.
- This is a must when contributing to Linux for the first time.
- This is also recommended even if you have prior experience, when contributing to a new subsystem for the first time

Early Research - Documentation

- The development process and the contribution process is well documented.
  - Documentation/development-process
  - Documentation/HOWTO

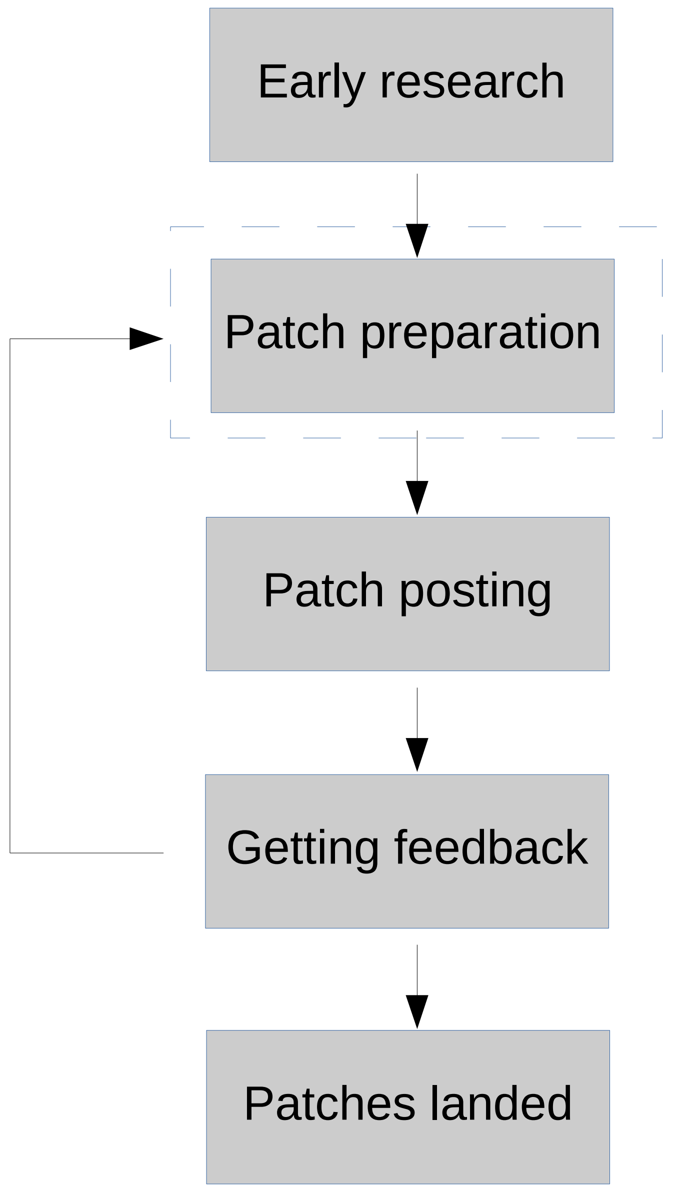
Early Research - Preferences

- Subsystems maintainers can have their own preferences.
- Learn the subsystem conventions for easier interaction.
- Look at the MAINTAINERS file to know who are the maintainers of a given subsystem.
- Search the subsystem mailing list archives for older threads to learn these unwritten rules.

Early Research - Preferences

- Some subsystems have their own documentation:
  - Documentation/devicetree/bindings/submitting-patches.txt
  - Documentation/networking/netdev-FAQ.txt
  - http://www.linuxtv.org/wiki/index.php/Development:_How_to_submit_patches
- Learning these preferences can feel like wasted time, but it really pays off in the long run.

Contribution Steps



Patch Preparation - Format

- Make sure patches conform to the canonical patch format.
- This is also very well documented.
  - Documentation/SubmittingPatches
- git format-patch
- Check the git log to use a proper subject line
- Include Certificate of Origin (Signed-off-by)
  - http://developercertificate.org/

Patch Preparation – Changelog

- Good commit messages explain **why** a change is needed, not **what** is changed.
  - The patch contents can answer what but not why
- What is in the changelog ends in the git tree
- Comments not suitable for the changelog should be included between a “---” marker line and the actual diff

Patch Preparation – Changes Split

- Split the changes in reasonable chunks so they can be reviewed easily.
- Patches should do only one thing, each logical change should be separated.
- Patches that can be grouped logically, should be posted as a patch series.
- A patch series should have a specific purpose.

Patch Preparation – Changes Split

- Patch series should not do too many things at once, it's better to split.
- Patches in a series should be added to be applied incrementally.
- Individual patches should not break bisect ability (for both build and run time).
- If a series contains fixes, these should be first. This allows them to be applied even if there are discussions about the other patches

Patch Preparation – Cover Letter

- Patch series should have a cover letter (PATCH 0/N) that explains what the series is about, how it was tested, etc.
  - git format-patch --cover-letter
- The cover letter should explain the dependencies between the patches and which patches should be applied by whom.

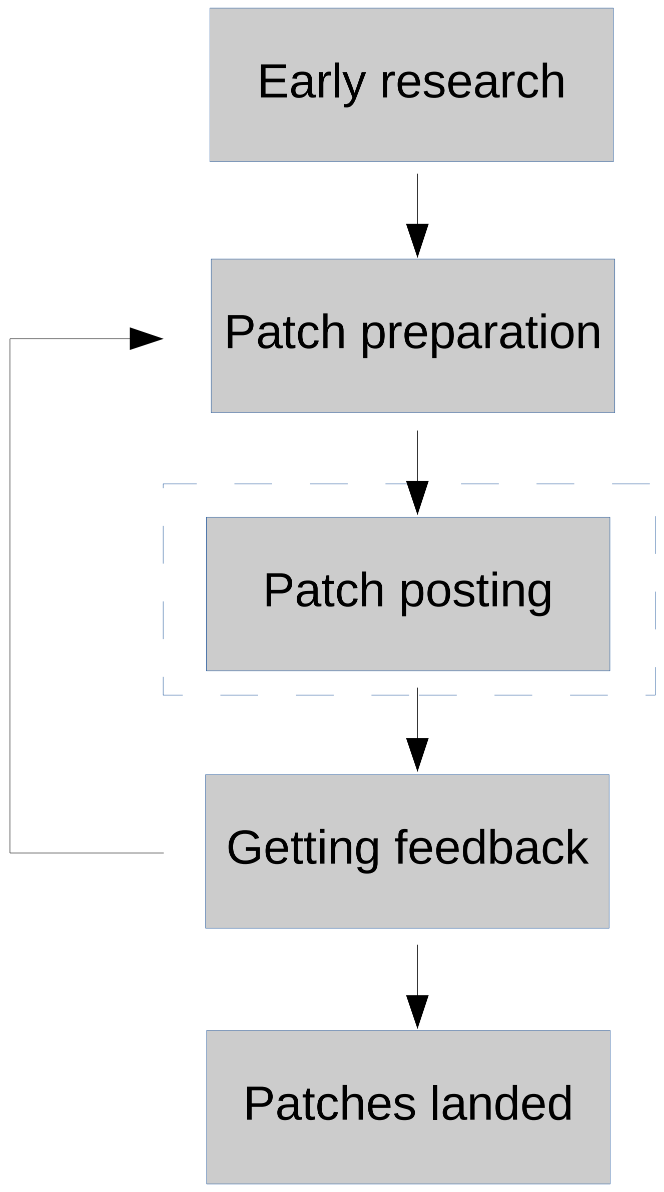
Patch Preparation – Dependencies

- If possible, all patches should go through the same tree.
- Or, let Kconfig handle the dependency (i.e: A depends on B).
- Make it clear if there are cross subsystem dependencies and indicate what these are.
- Cross subsystem dependencies (different ways to solve the conflict)
  - Split by kernel releases
  - Get Ack from maintainers and push everything through a single tree
  - Shared immutable branches between maintainers containing the dependencies patches

Patch Preparation - Tools

- git format-patch
- ./scripts/checkpatch.pl
- coccinelle
- sparse
- smatch
- cppcheck
- git rebase -i –exec

Contribution Steps



Patch Posting

- Documentation/SubmitChecklist
- Use git send-email since it does the right thing.
- If not sure about the patches, add RFC to the patches subject.
- However, some maintainers are too busy to look at RFC, so investigate their preferences.

Patch Posting – Who to CC

- It’s important to think about who should receive the patches and who shouldn’t.
- The MAINTAINERS file tells the maintainers and mailing list to send the patch to.
- The get_maintainer.pl script suggests a cc list.
- This it’s only a suggestion, don’t follow it blindly.

Patch Posting – Who to CC

- The decision to copy all patches in a series to all recipients is made on a case by case basis.
- Some people don’t like to be copied on random patches.
- Others prefer to get the entire series to have more context.
- Research the maintainers preferences to see what fits better with their workflow.

Patch Posting – CC'ing Cover Letter

- For patch series, the cover letter should be sent to all people receiving the patches.
- This way, everyone will have enough context to understand the patches.

Patch Posting – When to Post

- Maintainers also have different preferences on when patches should be posted.
- Some maintainers expects submitters to follow the development process, i.e:
  - Only post bug fixes during the -rc cycle
  - New features must not be posted during the merge window
- Other maintainers don't expect developers to know the dev process and picks both fixes and new features at any time.

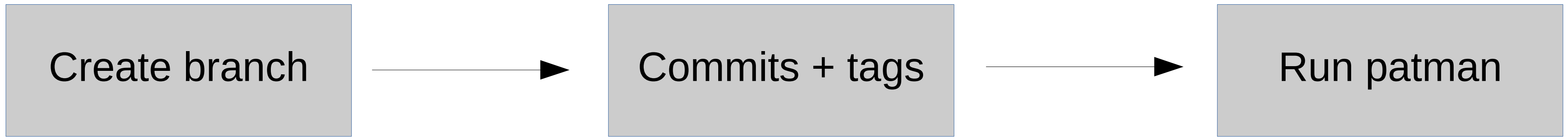
Patch Posting – Patman

- Developed by Simon Glass for the u-boot project
- Tool to automate patch formatting, check and submission
  - http://git.denx.de/?p=u-boot.git;a=blob;f=tools/patman/README
- Useful for any projects where the submission process includes posting patches
- Converts a git branch in a set of patches and post them

Patch Posting – Patman

- Behavior controlled by a set of tags in the commits
- Creates cover letter, logs, etc from metadata
- Invokes checkpatch.pl to verify the patches
- Calls get_maintainer.pl to fill cc list (or use tags in commits)
- Supports dry run option to simulate what would be done

Patch Posting – Patman Workflow



- For each patch series revision, the output will be consistent
- Reduces an unnecessary source of errors and annoyances versus when it's handed manually

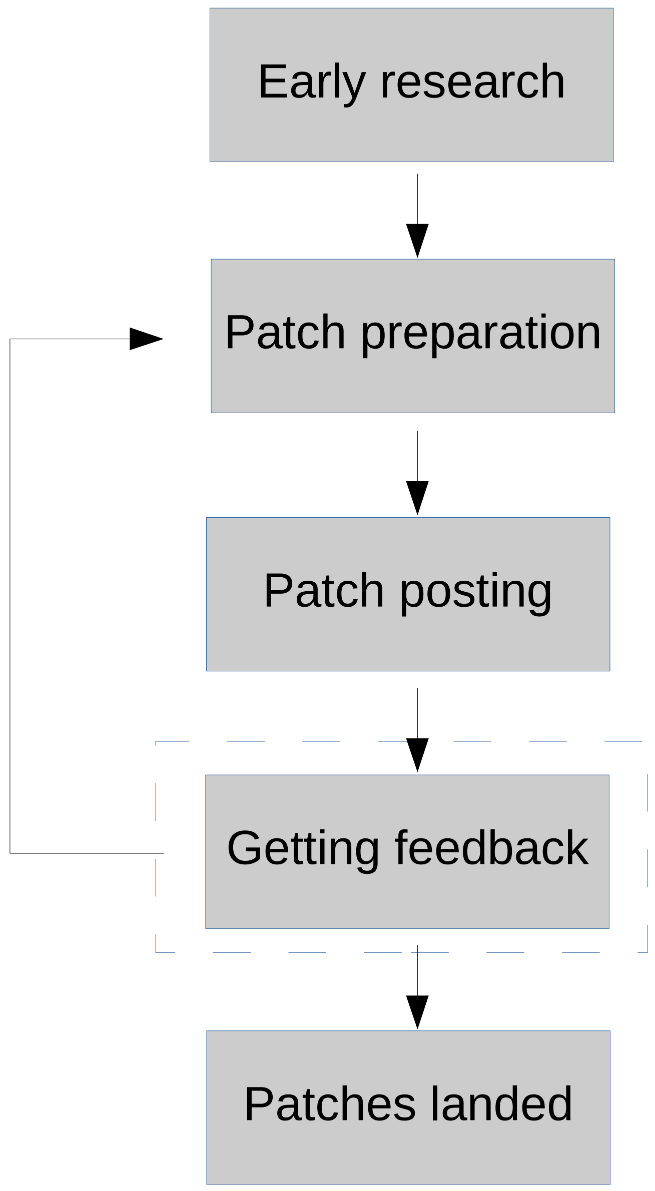
Patch Posting – Patman Tags

- Series-to: email address or alias to send this patch series
- Series-cc: email address or alias to copy this patch series
- Series-version: set the version of the series. Will add a v<n> to the patch subject
- Series-prefix: Set the patches prefix (i.e: RFC or RESEND)
- Cover-letter: Content of the cover letter, fist line is the subject
- Cover-letter-cc: email address or alias to copy the cover letter
- Series-changes: Changelog for patch series revision
- Commit-notes: Notes for each commit, appear after “---” cut
- Patch-cc: email address or alias to copy this patch

Patch Posting – Patman Options

- patman command line arguments
  - patman -n (dry run)
  - patman -c<n> (use the n first commits)
  - patman -s<n> (skit the first n commits)

Contribution Steps



Getting Feedback – Asking for it

- Give maintainers at least a week to answer.
- Some expect more time, so research their preferences.
- After a reasonable time, an action could be taken:
  - Some maintainers expect you to ask in the patch thread
  - Others maintainers expect the patch to just be resent
- This may depend on whether or not the subsystem uses patchwork.

Getting Feedback – Answer Inline

- **Don't top post**! Always answer the emails in-line.
- When discussing your patches remove unnecessary context from the email.
- People don’t want to scroll hundred of lines to read an answer of a couple of lines.
- But keep enough context so people answering after some days or weeks, can remember what the discussion was about.

Getting Feedback – Patch Revisions

- After feedback has been addressed, a new revision should be posted.
- A version v<n> should be included in the subject (i.e: [PATCH v2]).
- git format-patch -v2
- A log of the changes should be added between “---” and the diff.

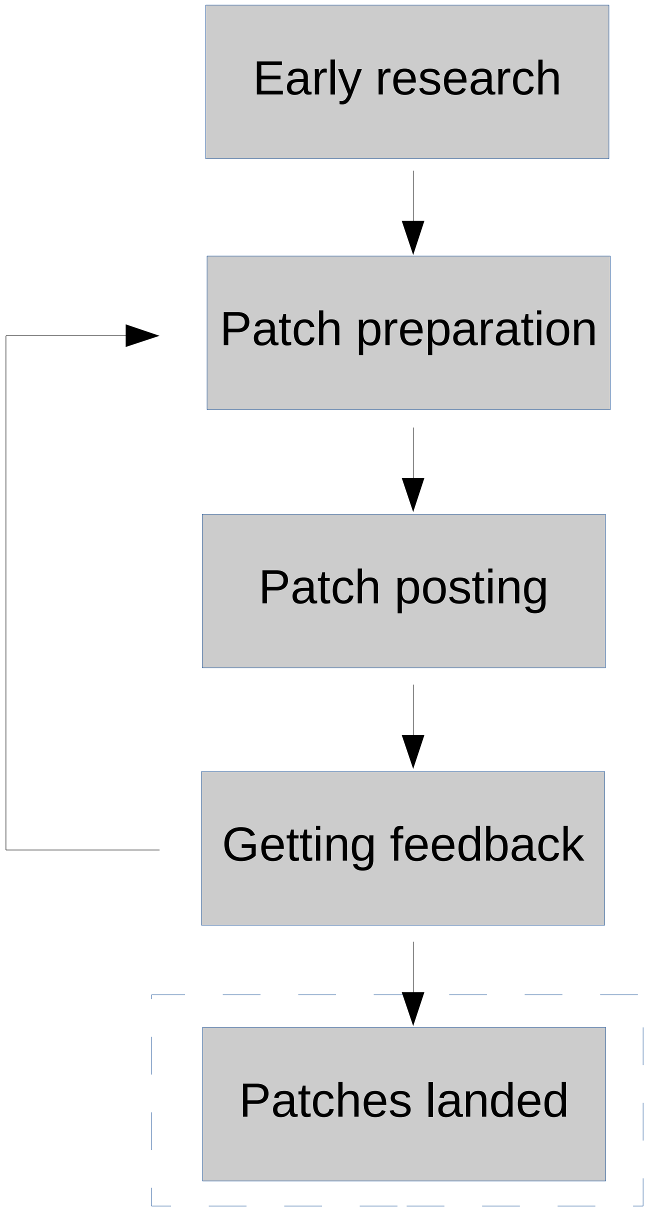
Getting Feedback – Patch Revisions

- Patches that have been ignored and are resent should have a RESEND prefix
- git format-patch --subject-prefix="RESEND PATCH"
- If a new patch is added to a series, mention it in the changelog.
- Patman makes all this easy (Series-version, Series-changes, Series-prefix).

Getting Feedback – Sending a new version

- Wait some time before sending a new version.
- It's possible that maintainers didn't have time to review yet.
- Sending too quickly could create more work for them.
- But could be that maintainers are not answering because a new version is coming.
- Again, this could depend on the maintainer so research the preference.

Contribution Steps



Patches Landed

- The work is not done when patches get merged.
- Patches will get a lot of manual and automated build & boot testing (kernelci, 0-day, etc).
- Make sure to be responsive in a timely manner if issues are found.
- Don't post patches and then disappear if bugs are found after merging.

Patches Landed

- Open source is about trust and this has to be earned.
- Maintainers expects submitters to be trustable.
- If that's not the case, they will be less fond to merge patches in future.
- It can affect the reputation of both the developer and the company they work for.
- So keep an eye to the subsystem you contributed and be ready to fix issues if these are found.

Questions?

Thank You!