



Efficient Kernel Backporting

Alex Shi

CLK Oct 22 2016

<http://www.linaro.org>



Agenda

- Why do feature backporting
- Before backporting
- Feature organization
- Deal with Conflicts
- After backporting
- Expectations on upstream



Why do backporting



Why use old LTS

- Time to market product want kernel:
 - Best stable kernel
 - A target kernel as system baseline, for
 - Out of tree patches
 - System compatibility
 - Less regression testing required

A steady LTS



Feature Missed in LTS

- Latest feature missing
 - Latest drivers, functions or security updates, like
 - Arm64 PCI-E support on 3.18/4.1
 - Cgroup writeback, hibernation on arm64, KASAN on 4.1
 - Or out of tree features:
HMP/EAS on ARM for big LITTLE arch.



Solutions and Disadvantage

- Solution: LTS + feature backporting
 - An example: Linaro Stable Kernel
 - <https://wiki.linaro.org/LSK>
- Perfect solution? No
 - Repeated work for enabled feature
 - Less review/testing from community
 - Trouble with newer API or coupled kernel components



Other Solutions?

Keep rebasing on upstream kernel?

- Less stability
- kernel API change, like: driver API, /proc, /sys, ioctl
- Keep regression testing



Best Feature Candidate

- Good resources to know request feature
 - Get feature info from requester, what/why
 - Get feature profile from lwn.net or wiki
- Know the feature versions
 - First target is always the upstream patchset
 - An old version feature maybe acceptable, if it's using old kernel API



How to do Backporting



Get Feature Commits

- Get feature patchset from lkml
- Collect feature commits in git tree
 - From related source files and headers
 - `Git log v4.1..upstream - drivers/base/power/opp`
 - If feature name mentioned
 - `git log -i -G'kasan' v3.18..upstream` (54 commits)
 - `git log -i --grep=kasan v3.18..upstream` (126 commits)
 - By involved authors
 - `git log --author=tj@kernel.org --reverse v4.1.18`



Feature Organization

- Rules for feature Organization
 - Each of feature isolated in a separated branch
 - Dependent feature based on others branch instead of LSK mainline
 - Tree mode branches
 - A example: pax_usercopy rely on KASLR
- Benefit
 - Feature Selectable
 - Reading, retrieve and debug



Feature Organization

- An example of LSK organization:

```
-----LTS 4.1.17 ----- LTS 4.1.18
      \                   \
----- LSK 4.1 16.02 ----- LSK 4.1 16.03
      /                   /
---LTS 4.1.17+ FooA      /
      \                   /
      FooB-----
```



Backporting with Git

- Pick up wanted changes/commits

`$git cherry-pick -sx commit1..commit2`

- git cherry-pick a patchset range
 - Get all commit at once, easily disturbed by dependent missing.
- git cherry-pick patches one by one.
 - Easy to control for every commits



Conflicts

- Middle changes missed conflicts
 - Direct code base changed
 - Like, removed line doesn't exist.
 - Know reasons of the changes by 'git log -S/Gstring', pick or skip
 - Miss some dependencies,
 - Like some new code line in close section
 - Find and pick them by 'git blame' or 'git log -p'
 - Feature coupled with other kernel components
 - Cut off the connection maybe better
 - Or bring trouble on the other kernel part



Tool for Conflicts

- Conflicts solution reuse
 - Use git rerere to reuse conflicts solutions
- Find commits that trigger conflicts
 - Tools git log, git blame
 - `git log -G'extern void inet_twsk_put' - include/net/inet_timewait_sock.h`
 - `git blame - drivers/base/power/opp/core.c`



Reduce Conflicts

- Reduce conflicts
 - Find out all related changes on feature related field
 - Pick up from base to upstream direction
 - Pick up the related set commits instead of only single related patch,
 - Use gitk not git log --graph to find out the commit set



Feature/LTS Conflicts

- Example, a upstream situation:

other blk/mm change
/
----- 4.1 -- bug fix f1 ----- \ conflicts -- v4.2 ---- bug fix f2/f3 ----
\
cgroup feature -- a -- b /



Feature/LTS Conflicts

- Usual solution for cgroup writeback LTS branch:

```
                backported cgwb -- a' - b'-- c'
                /
--- 4.1 -- 4.1.x ---- 4.1.18
                /
            bug fix f1/f2/f3
```



Feature/LTS Conflicts

- A better way, postpone conflicts to merge
 - Do the backporting on v4.1 kernel, and left the conflicts on merging to 4.1.18

```
      backported cgwb -- a -- b
      /                          \
--LTS 4.1 -- 4.1.x ----- 4.1.18
      /
bug fix f1/f2/f3
```



Don't Do ...

- Don't change kernel/user API /sys, /proc etc
 - System libs, applications rely on them
- Don't pick up big coupled kernel parts
 - Cut off early
- Don't change driver API
 - Downstream drivers



Post Backporting

- Review if all necessary commits picked
 - Scan and compare all changed and related commits

```
git log --cherry v4.4...v4.1/topic/KASAN -- ./mm/kasan
./include/config/have/arch/kasan.h ./include/linux/kasan.h
./arch/arm64/mm/kasan_init.c ./arch/arm64/include/asm/kasan.h
```
- Testing for the feature
 - Seek testing method from community



Post Backporting

- Scan bug fix for picked commits

<https://git.linaro.org/people/alex.shi/scripts.git/blob/HEAD:/chkfix.sh>

The latest features are often buggy, so scan the upstream kernel, to see if any commits which you picked was mentioned by others, that's probably a bug fix.



Post Backporting

- Notice your users of Any API changes if you have to do
 - Explain reasons
 - And give compatible solution for changes



Summary

- Why Backporting is a common needs in industry
- How to do backporting and how to resolve conflicts
- Standardization needs on upstream



Expectation for Upstream

- Stable
- The less API change, the better.
 - Standardization of API?
 - POSIX, Libcgroup, DRM, memfd, etc.
 - Driver API
 - keep old function when new ones introduced
 - Good examples, cgroup v2 coexist with cgroup v1
- Collaboration on backporting
 - LTSI/LSK



Summary

- Why Backporting is a common needs in industry
- How to do backporting and how to resolve conflicts
- Standardization needs on upstream



Thanks!