LAS16-101 - Efficient Kernel Backporting

Alex Shi
Agenda

- Why need do backporting on old kernel
- Preparations of backporting
- How to do Backporting
- Conflicts & solution
- Post backporting
- Expectations on upstream
Why is old stable kernel?

- Product considerations
  - Product Time line, life cycle
  - Kernel/user API, in-kernel API
    - /sys, /proc,
    - Driver API
  - Stability
Extra Features Needed

● Desire for extra features?
  ● 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](https://wiki.linaro.org/LSK)

- **Perfect solution?**
  - Repeated work for enabled feature
  - Less review/testing from community
  - Trouble with newer API or coupled kernel components
Ideal Solutions?

- A ideal upstream kernel?
  - A bug free kernel? no
  - Consistent kernel API, like: driver API, /proc, /sys, ioctl? no
Before Backporting
Before Backporting

- Get requested feature resources:
  - 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
Get all related commits

- Know feature patches from lkml
- Get commits in git tree
  - Get commits list from feature related source files and headers
  - All commits mentioned feature name
  - Get all writeback cgroup patches by author

```
git log -i -G'KASAN' v3.18..v4.4 (54 commits)
git log -i --grep=kasan v3.18..v4.4 (126 commits)
git log --author=tj@kernel.org --reverse v4.1.18.
```
Backporting
Backporting

- 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
Backporting Path

- An example of backporting organization:

-----LTS 4.1.17 ----- LTS 4.1.18
    \  \                      \  \
--------------- LSK 16.02 ---------- LSK 16.03
      \                                 \
---LTS 4.1.17+PAN            /  \
                                      /  \
-------------------LTS 4.1.18 + Cgroup WB
Conflicts & Solutions
Conflicts & Solutions

- Conflicts: middle changes missed
  - Direct code base changed
    - Know reasons of the changes, pick or skip
  - Miss some dependencies
    - Find and pick them
  - Feature coupled with other kernel components
    - Cut off the connection maybe better
    - Or bring trouble on the other kernel part
Feature/LTS Conflicts

- Possible upstream situation:

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

- 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: Do the backporting on v4.1 kernel, and left the conflicts on merging to 4.1.18
  backported cgwb -- a -- b
  /-----------------------\
  --- 4.1 -- 4.1.x ---------- 4.1.18
  /                           
  bug fix f1/f2/f3
Tips for Conflicts

- Reduce conflicts
  - Find out all related changes on feature code area
    - 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
Tips 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 log v3.10..v3.18 -S'newsk->sk_v6_rcv_saddr = ireq6->loc_addr' --net/dccp/ipv6.c
Don't Do ...

- Don't change kernel/Use API `/sys, /proc` etc
  - System applications
- Don't pick up big coupled kernel parts
  - Cut off early
- Don't change driver API
  - Downstream drivers
Post Backporting

- Check if all necessary patches 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
  - 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](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 so.
  - Explain reasons
  - And give compatible solution for changes
Expectation for Upstream

- Stable, stable, 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

- More collaboration on backporting
Thank You

#LAS16

For further information: www.linaro.org
LAS16 keynotes and videos on: connect.linaro.org