Managing multiple android kernel trees (with quilt)

Mark Gross

Mark.gross@intel.com

March 2015

Outline

Problem definition and scope
Strategies
Quilt use tutorial

Key points

Decouple Android integration and kernel development:

- Binary kernel delivery into Android builds
- Only kernel team gate keeps kernel code.
- Only kernel team uses quilt, other contributors must use normal git trees and gerrit.

Engineer support for multiple kernel versions per user mode binary.

- Binary patching of DUTs with updated kernels is the test methodology to strive for.

Actively organize and annotate quilt series as development progresses.

Quilt use has not been a bad experience for the OTC Android kernel team.

- Just limit quilt use by non-kernel team personnel.

Problem definition

Deliver multiple tested kernels with variants into android lunch targets a few times per day
Efficiently review and integrate kernel contributions from a large number of engineers
Prevent regressions
Keep the many consumers of Android Linux kernels “not angry”

Keep kernel change bill of material well organized
Communicate technical debt and help drive selected upstreaming

Android Kernels at Intel

Intel currently has the following kernels in play for different SOCs for android-L

- 3.10 : used in the Fugu and Dell venue 8 7000
- 3.14.y : used in BYT tablets
- 3.14.y : used in SoFIA devices
- 3.19 : targeting a newer SOC working through its power on
- Dev : tracks upstream (currently 4.0) to be used for pre-Silicon

We also have “debug” variants for each of these kernels used in –eng targets.

- with additional changes

Strategies

Strategies in use today
(Don’t hide complexity.)

Long-lived branches are used for stable release sustaining only.

- Separate kernel trees (quilts) are created rather than using branches for different SOCs.

Only use full kernel config files (diff-configs are bad).

Organize your kernel change BOM actively.

Report your kernel technical debt regularly.

Strategies in use today (moderate kernel regression testing)

Binary patch test methodology for kernels onto DUTs

- Avoid rebuilding Android for each kernel test.

Implement a no-obvious-regressions-allowed integration and test policy.

- Kernel releases are regression-tested against the latest user mode build by kernel team QA before release into Android integration.

Strategies in use today (drive modularization)

Binary kernel delivery into Android integration builds

- Kernel changes are managed exclusively by kernel team.
- Integration teams can choose to take a kernel release or not.
- Avoids conflicts of interest between feature teams and integration teams.

Use signed kernel modules

- Delete the key after each build to prevent security issues.

Strategies in use today (drive scalability)

Drive scalable binaries whenever possible. (BYT-CHT share a binary)

- But don't push too hard too soon or you'll have a mess of a kernel BOM.
- Drivers need to coexist with multiple platforms at a binary level.

Driver ABIs need to scale from Nexus design to "feature rich" or "differentiated" stack.

- Need to support standard HALs as well as fancy new HALs gracefully

Enforce ABI compatibility from kernel and user mode.

- Always need to be able to test new kernels with recent user mode images.

Strategies in use today (use quilt)

Use Quilt to manage kernel software BOMs.

- Organize the patches logically.
- Minimize the series for each kernel.

Don’t let driver developers use quilt.

- Make them use git and gerrit.
- Git quiltimport is your friend.

Quilt use within the OTC Android Kernel Team

Tutorial section

Quilt use

Keep your quilt series in a git project.

Use a symbolic link to the patches directory in a separate git project from the working Linux tree.

Generate patches using git format-patch run in a separate Linux tree and add them to your quilt by hand.

Add the patch to a logical location in the series file.

- Quilt graph helps with this.

Our approximate kernel source SCM structure (demo)

Quilt-representation.git/

- Bin ← holds updated patch program, mingzip, packageandprovision.sh
- Assorted build scripts (one per quilt tree)
- Uefi/production/
  - X86_64.config
  - I386.config
  - Modules/ ← external modules
  - Patches/
    - Series
    - *.patches

Repeats pattern for each kernel

…

Quilt implementation details (demo)

Quilt implements a stack structure with within the .pc directory where it holds the data needed to push and pop the current patch in the series.

Core quilt data structures:

- patches directory with patches and series file
- .pc directory
  - holds pre-patched copies of files in subdirectories directories named after the patch files
  - Holds ordered list of currently applied patches.

I use the following hack to quickly reset my working tree:

- “git clean –df; git reset –hard; rm –rf .pc”

Quilt traps

Forget to add files to patches

- Quilt will not add a file to a patch implicitly until at least one hunk applies, i.e. your fix-ups can be lost.
- Use quilt edit.

Adding a file to the patch after you changed them

- Use quilt edit.

Nuking your work within a git working tree with “git clean –xdf”

- Use a sym-link to the patches kept in a separate git project.

Old patch program can’t deal with symbolic links (need version 2.7.x or newer. Ubuntu 12.04 version of path will not work.

Breaking git quiltimport (-- missing author)

Common quilt commands

Quilt push

- Quilt push -a
- Quilt push -f

Quilt pop

- Quilt pop -a

Quilt new

Quilt add

Quilt refresh

Quilt delete

- Quilt delete -n

Quilt graph <-- the most awesome command ever

Quilt graph is my jam (demo)

Quilt graph will provide a dependency tree of all patches related to the current “top” patch.

- Enables simple ordered clustering of changes by touched files.

- Boom.

My quiltrc

```
QUILT_PATCH_OPTS="--fuzz=0"
QUILT_PUSH_ARGS="--color=auto"
QUILT_DIFF_ARGS="--color=auto --no-timestamps --no-index"
QUILT_SERIES_ARGS="--color=auto"
QUILT_PATCHES_ARGS="--color=auto"
QUILT_REFRESH_ARGS="--diffstat --no-timestamps --backup --no-index"

QUILT_DIFF_OPTS="--show-c-function"
```

Work flow recipes

Quilt fold ~= git rebase squash

1. quilt graph patches/FOO.patch to see the first patch that FOO.patch depends on.
2. Edit patches/series file and move the line "FOO.patch" to the line following the patch identified by quilt graph.
3. While editing insert "stop-and-fold-1-random-non-comment-line-string" on the line above FOO.patch.
4. Reset the git tree: git clean -df; git reset --hard; rm -rf .pc
5. quilt push -a
6. quilt del -n; # removes the stop-and-fold-1 maker
7. quilt fold < patches/FOO.patch
8. quilt del -n; # removes the FOO.patch you just folded into the previous change.

Rebase work flows
(Step 1: Fix Conflicts)

While not done patching:

- Quilt push -a
- Quilt push -f
- Quilt edit <file> <file>.reg patches/<current patch>
- Quilt refresh
- Quilt push -a

Rebase work flows
(Step 2: Fixing Compile)

While not yet compiling:

- Clean and test compile
- Quilt new fixup-<filename-with-xyz-compile-issue>.patch
- Quilt edit <filename-with-xyz-compile-issue>
- Quilt refresh

Rebase work flows
(Step 3: Organize Series)

For each fixup-<xyz>.patch

- Quilt graph fixup-<xyz>.patch
- Edit series file to relocate fixup-<xyz>.patch.
- Fold fixup-<xyz>.patch if it makes sense.

Rebase work flows (demo)
(Step 4 : quiltimport fixing + check in work)

Test and fix git quilt-import issues (demo)

- Note patches that you have to fix as you run the git command.
- Do a git format-patch after fixing all the quilt-import issues.
- Copy the noted patches (now fixed) over the bad ones in the patches directory.

Check in changes to quilt series added patches.

Gerrit patch harvesting work flow (demo)

Starting with the git quiltimport created git project used by non-kernel teams:

- Use normal gerrit interface to cherry-pick the patch into the working git tree.
- Use git format-patch to create the patches.
- Copy patches into quilt-representation.git project
- For each patch:
  - Add patch to tail of series file.
  - Apply series to test kernel.
  - Organize series -- run quilt graph to find a good location for the patch.
  - Edit series file relocating the patch in a better location.
  - Annotate series file.

Quilt features I'd like to get:

Git quiltimport to recreate the same SHAs when run at different times.

Quilt patch squashing interface closer to git.

Quilt patch splitting

Make quilt a little more idiot-proof overall.

Key points

Decouple Android integration and kernel development:

- Binary kernel delivery into Android builds
- Only kernel team gate keeps kernel code.
- Only kernel team uses quilt, other contributors must use normal git trees and gerrit.

Engineer support for multiple kernel versions per user mode binary.

- Binary patching of DUTs with updated kernels is the test methodology to strive for.

Actively organize and annotate quilt series as development progresses.

Quilt use has not been a bad experience for the OTC Android kernel team.

- Just limit quilt use by non-kernel team personnel.

The End

Thank you!

Assorted Execution Details (optional to talk)

CI + testing

We use Jenkins to run CI that includes:

- Aiaiai static analysis checks
- Staging builds to feed automated testing

Automated BAT tests are run as part of CI

- Some LTP
- Some CTS
- Some “other tests”

Light user tests are done on release candidates prior to release into Android Integration.

PackageAndProvision tool
Binary kernel patching

Works on user-debug and eng builds (you need root).

Needs to disable dm-varity to update kernel modules in system partition.

- “adb disable-verity”
- Would be nice if kernel modules were in a separate partition (or ramdisk).

Takes existing boot.img, cracks it open, inserts a new kernel, re-packs it, and signs it.

Adb remount system partition and push new *.ko files

Adb reboot bootloader

Fastboot flash repacked boot.img

Reboot.

Test.

Tool chain for kernel

I wanted to decouple the kernel tool chain from AOSP and host distros with a built-from source solution.

I chose the Yocto tool chain created from the 1.6 release

- Git clone poky
- Git checkout -b daisy
- Source poky/oe-init-build-env toolchain
  - Edit config/local.conf ← change target to match your HW
  - MACHINE = qemux86-64
- Bitbake meta-toolchain

TARGET_TOOLS="/opt/poky/1.6/sysroots/x86_64-pokysdk-linux/usr/bin/x86_64-poky-linux”
CROSS_COMPILE="$TARGET_TOOLS/x86_64-poky-linux-“

Note: I will upgrade to the next yocto release soon.