Reducing the pain of Yocto development upgrades

Michael Brown – NGM Firmware Lead Technologist – Dell EMC

Embedded Linux Conference 2017

Outline – Easier Yocto upgrades in development

Who am I?

Michael Brown – Dell EMC

I am a 17 year veteran of Dell. During that time, I have done a lot around build architecture. Most recently I led the yocto-ization of the Dell EMC Firmware builds, porting the build of the Dell IDRAC and CMC from a hand-rolled monolithic build over to a fully-componentized Yocto build. I designed everything from individual component autotools layout up to the entire Yocto layout.

I am currently the lead technologist for embedded management on our next generation chassis.

Eventually they are going to figure out that they never revoked my git admin access when I turned over all those servers to our official build team.

Problem Statement

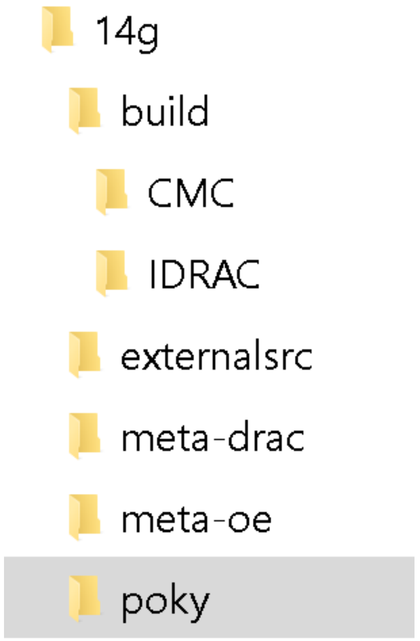
- I’m going to talk about keeping development environment current, NOT updating devices
  - Often not addressed. The obvious way to do this is often the hardest and riskiest.
- Keeping development up to date is hard!
- Breakages are tough in development and after a few, managers and leads get gun shy
- Worst case scenario is development can be broken, stopping entire development teams
- On the other hand, delaying updates can be bad: security issues, interlocking dependencies, etc can make it hard to do piecemeal updates. Getting a high severity defect patched quickly can be impossible. How do you update that 2.4 kernel, glibc 2.1, and gcc 3.2? (um, asking for a friend…)
- I’m giving this talk specifically addressing Yocto, however the concepts can be applied in many environments

Dell EMC firmware development environment

- This update strategy has been in use on the Dell EMC firmware development team for 2 years now. We've done 4 major Yocto updates using this method.
- Dell EMC firmware team is large and encompasses:
  - IDRAC (Embedded Server Management) for 12G and 13G servers
  - Chassis Management Controller (CMC) for M1000e and other chassis.
  - IDRAC for our in-development servers and CMC for our in-development chassis
  - **IDRAC and CMC codebases built from one Yocto environment starting with our next generation servers**
- Dell EMC Yocto environment:
  - Base Yocto environment: Poky plus select meta-oe components.
  - Roughly 300 Dell EMC components.
  - Each component is a standalone GIT repositoriy with a standalone Autotools build producing a library or set of binaries
  - Tens of thousands of commits across these per release

Development Setup

- Android "repo" tool: https://source.android.com/source/using-repo.html
  - We have about 300 repositories to check out to do a build
  - Full from-source build
  - Highly recommend versioning everything identically: same branches, tags, etc everywhere
  - Our branch naming scheme:
    - rel/14g/master
    - rel/14g/1.0/master
    - rel/14g/1.1/master
    - Hierarchal namespace to sort tags and branches
    - Never use "master" because it is very difficult to use external git repos that also use "master"



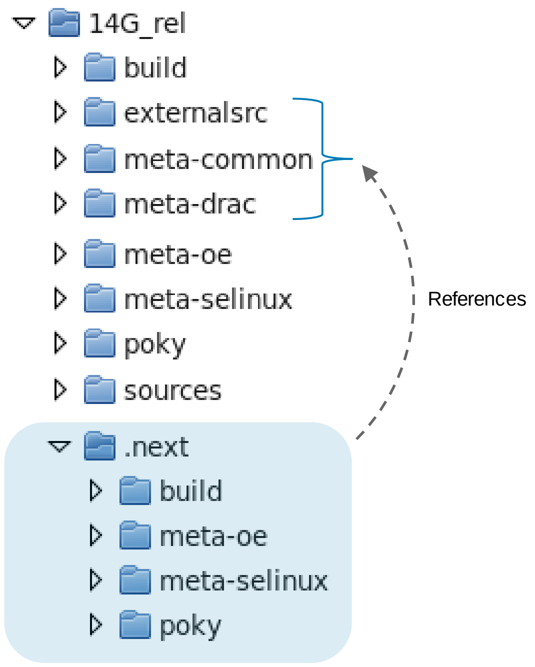
The Big Idea

- The core of this method is to have an extra build called “poky-next”.
  - (cue audience gasp: we waited through 4 slides to hear *this*?)

- Yes, this is really basic, but it seems to be nonobvious.
- Here are the core requirements/ideas
  - The poky-next build is a **parallel** build structure. You can build either using regular poky, or with poky-next
  - Builds **your same** source code as the “normal” build (for all non-yocto components)
  - **Small** units of work: update frequently so that each individual update is manageable.
  - **Separate** source control copy for the poky and poky-next repos so that you can carefully control the flow of updates into the tree

- Benefits:
  - Work on Hard Stuff ™ without breaking main development stream.

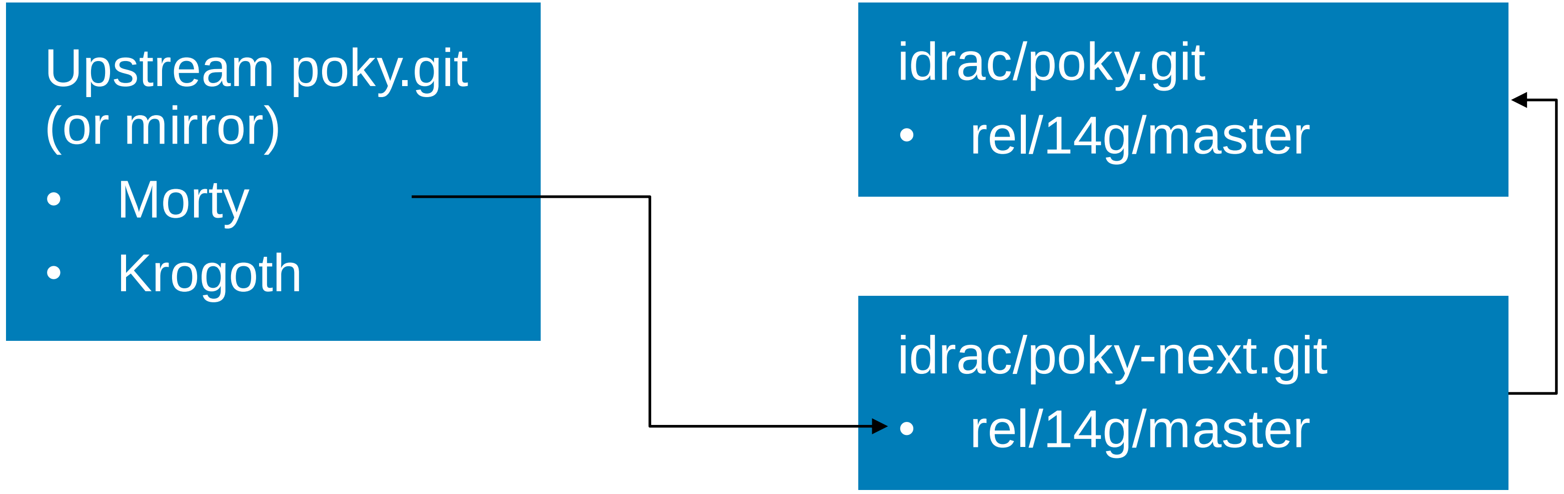
Directory Structure



- New directory ".next" is created.
- The .next/build/ directory config has bblayer files that reference poky, meta-oe, and other upstream meta layers under the .next directory
- The .next/build/ directory config bblayer files reference the main layer meta-drac and other local meta layers, however, for complicated cases, these can be branched as well

Git repositories

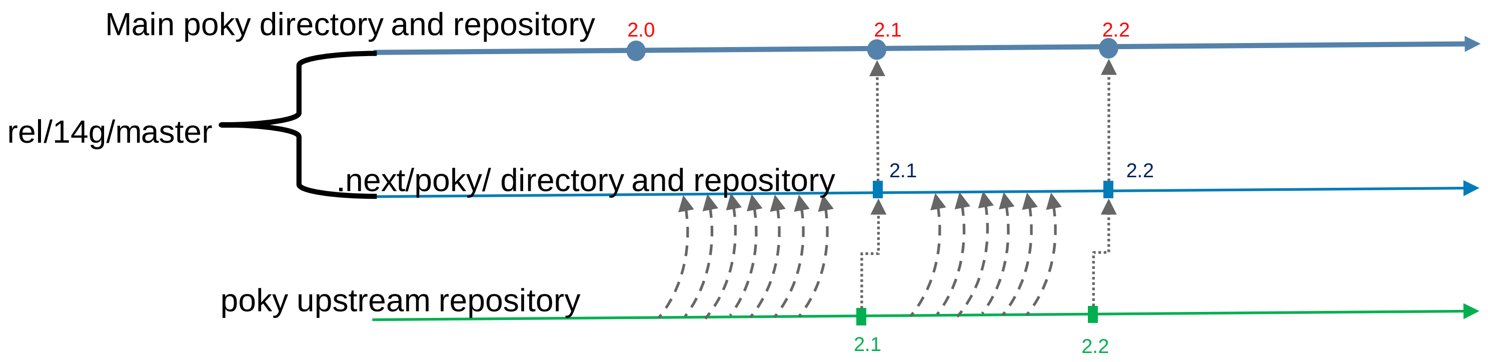
Our design has everything on the same branch, so we have multiple repository copies that have our branch (re/14g/master) tracking different upstream branches (Morty, Krogoth, etc). You could easily design something similar with one repository and multiple branches.



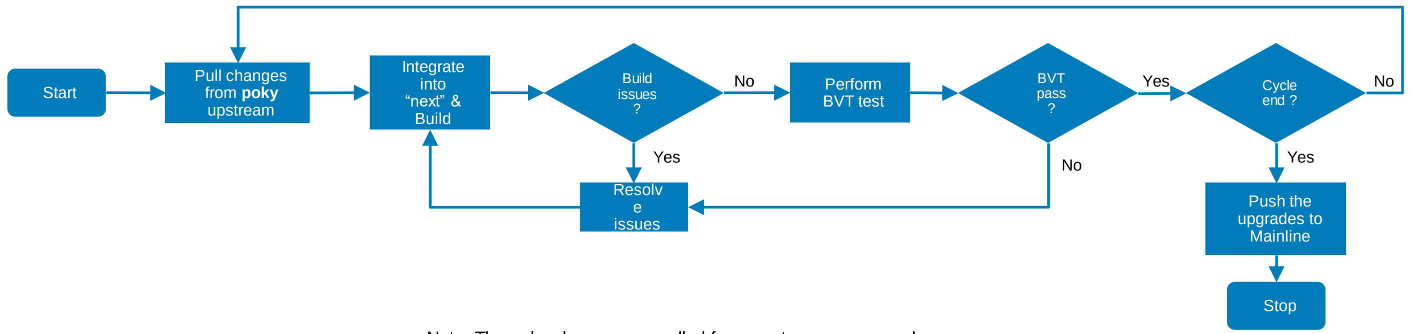
Repository Manifest

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <remote  name="origin" fetch=".." />
  <default revision=“rel/14g/master" remote="origin" sync-s="true" sync-j="4" sync-c="true" />
  <project path="poky" name="idrac/poky.git" />
  <project path="meta-drac" name="idrac/meta.git" />
  <project path="build/configs" name="idrac/buildconfigs.git" />
  <project path=".next/poky" name="idrac/poky-next.git" />
  <project path="externalsrc/dell-emc-example" name="idrac/dell-emc-example.git" />
</manifest>
```

Branching Overview: poky-next


Main poky directory and repository
2.0
2.1
2.2
rel/14g/master
.next/poky/ directory and repository
2.1
2.2
poky upstream repository
2.1
2.2

Poky Next Branch


Start
Pull changes from **poky** upstream
Integrate into "next" & Build
Build issues ?
No
Yes
Resolv e issues
Perform BVT test
BVT pass ?
Yes
No
Cycle end ?
No
Yes
Push the upgrades to Mainline
Stop

Note: The poky changes are pulled from upstream every week

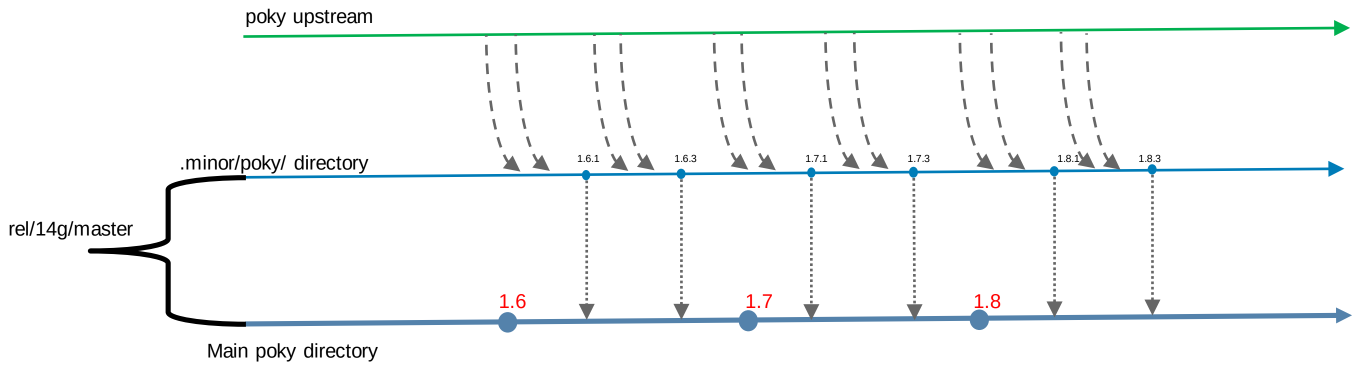
Benefits

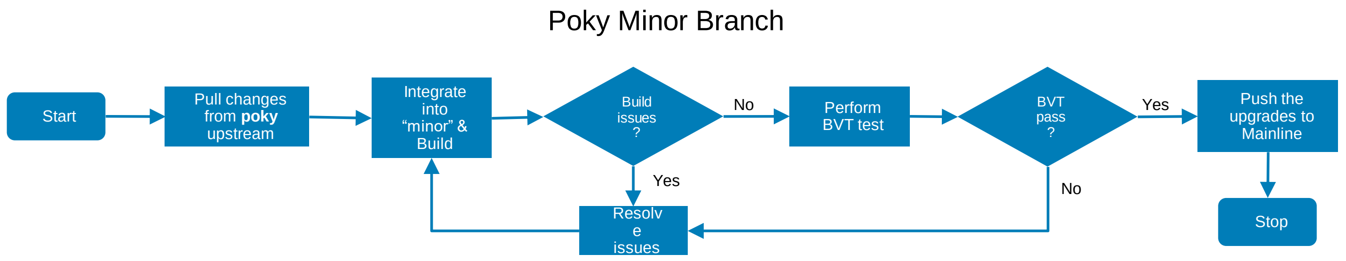
- Great for long-lived development projects
- Smaller units of work for doing updates
- Predictable
- Control over when updates go into production
  - Easier to work with scheduling disparities between upstream and your release schedule.
- Extensively testable – Jenkins (or equivalent) can do daily/continuous builds of the .next build.
- Less stuff breaks when you do smaller updates

Taking it further

- The .next concept works well for in-development releases to keep them up-to-date
- Once you have done a release, switch concepts to “**.minor**”. Instead of following upstream “master”, follow the upstream fixes branch.
- Even further: for long-lived released products, combine the .minor and .next concepts to keep devices completely up to date per release, and then migrate them from Yocto release to Yocto release

Branching Overview: poky minor




Poky Minor Branch
Start
Pull changes from **poky** upstream
Integrate into "minor" & Build
Build issues ?
No
Yes
Resolv e issues
Perform BVT test
BVT pass ?
Yes
No
Push the upgrades to Mainline
Stop

Note: The poky minor releases are pulled from upstream every 3 weeks

End of Slides – Demo Time