WORKING WITH THE LINUX KERNEL IN THE YOCTO PROJECT

SEAN HUDSON

EMBEDDED LINUX ARCHITECT

BIO

- Developing software for embedded devices since 1996
- Started using Linux in 1999 to create a NAT router for my DSL internet connection
- Developing embedded Linux devices professionally since 2006
- In the OSS world, I am Member Emeritus of the YP Advisory Board, a member of the OpenEmbedded Board, and part of the devicetree.org Technical Steering Committee

PRESENTATION OUTLINE

- Project advertising ☺
- Disclaimer
- Some Basics
- Important BitBake Kernel Tasks
- Basic Developer Workflows
- Live Example
- Additional Observations/Recommendations
- Questions

See a Yocto Project presentation today!

yocto PROJECT

MONDAY		
2:00pm	Pavilion West	Working with the Linux Kernel in the Yocto Project - Sean Hudson
3:00pm	Grand Ballroom 1	OpenEmbedded/Yocto on RISC-V - Khem Raj, Comcast
TUESDAY		
10:50am	Pavilion East	Comparing and Contrasting Embedded Linux Build Systems and Distributions - Drew Moseley, Mender.io
11:50am	Pavilion East	BoF: Yocto Project & OpenEmbedded - Jeffrey Osier-Mixon, Intel
11:50am	Pavilion West	Speeding your Linux Development with Debian and OpenEmbedded on DragonBoard 410c - Mark Charlebois, Qualcomm
3:00pm	Pavilion East	Real-World Yocto: Getting the Most out of Your Build System - Stephano Cetola, Intel
4:20pm	Pavilion West	poky-tiny and Beyond, or Trying to put the Yocto in Yocto Project - Scott Murray, Konsulko
WEDNESDAY		
11:05am	Broadway III	Learn Bitbake with Yocto Project - Tom King, Linux Foundation (E-ALE) (2hrs)
3:30pm	Grand Ballroom 1	Living on Master: Using Yocto Project, Jenkins and LAVA for a Rolling Release - Tim Orling, Intel
3:30pm	Galleria North	DIY Connected IoT Products using Open Source Software - Alan Bennett & Tyler Baker, Open Source Foundries
4:30pm	Grand Ballroom 1	Building Images with Yocto Project - Tim Orling, Intel (E-ALE) (2hrs)
THURSDAY		
all day	offsite	Yocto Project Developer Day, sponsored by Mentor Graphics - sign up now!

YOCTO PROJECT DEVELOPER DAY

- The Yocto Project is happy to announce Yocto Project Developer Day taking place at Mentor Graphics, Wilsonville Oregon, USA, on March 15th, 2018. This is the day following the Embedded Linux Conference being held at the Hilton in downtown Portland Oregon, so make your plans to leave a day later and learn more about Yocto Project's open source, high-quality infrastructure and tools. (Note that this venue is a 40 minute, complimentary bus ride from the ELC venue.)
- **Date:** Thursday, March 15
  Time: 9:00am – 5:00pm
  Location: Mentor Graphics (8005 Boeckman Road, Wilsonville, Oregon, 97070) *
  * Transportation will be provided from the Portland Hilton. Please plan to meet at 7:45am in the hotel lobby.
- **Registration Cost:**
  Late Registration: $249 (March 1 – Event)
- https://www.yoctoproject.org/learn/yocto-project-dev-day-north-america-2018/

YOCTO PROJECT BOOTH

- Talk to key developers and maintainers
- Get more information about other sessions
- Get more information about the Yocto Project Developer Day
- Get some free swag!
- Chance to win a free copy of Rudi's book

DISCLAIMER

- There is more to this topic than I can possibly present in this session. So, I am only going to give a quick survey. For more depth, I strongly encourage developers to attend a Yocto Project Developer Day, read the excellent project documentation (Thanks Scott Rifenbark!), check out "Embedded Linux Systems with the Yocto Project" by Rudi Streif.

SOME BASICS

NEEDED KERNEL INFORMATION

- In order to build a kernel, you will need the usual suspects:
  - Kernel source (with any patches)
  - Kernel .config
- In addition, you will need a recipe (metadata) to integrate into the build system

CLASSES OF KERNELS TO CONSIDER

- Linux-yocto kernels
  - These are kernels that work with the linux-yocto kernel tooling
- Traditional
  - These are kernels that are already available in meta-data layers, but do not use the linux-yocto tooling
- New
  - Kernel source from *somewhere*, most likely a semi
  - No specific tooling related to the Yocto Project

A BIT MORE ON LINUX-YOCTO KERNELS

- The Yocto Project maintains kernels and a set of tooling that facilitate supporting multiple platforms with a common kernel
- One of the key benefits comes from kernel configuration management
- These tools allow the developer to use ‘configuration fragments’ to control specific kernel options
- This is accomplished by following a specific git structure that enables the tooling to find and assemble a complete configuration for a kernel
- This feature is highly configurable and, therefore, highly complex to setup
- Luckily, the Yocto Project maintains several of these kernels for you

EXAMPLE RECIPE TO ADD A KERNEL TO THE BUILD

- Simplest approach just adds the kernel source and configuration into the build
- Relies on the kernel.bbclass
- Trimmed down, real world example*

```
SECTION = "kernel"
DESCRIPTION = "Linux kernel for TI devices"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = file://COPYING;md5=d7810fab7487fb0aad327b76f1be7cd7
inherit kernel
S = "${WORKDIR}/git"
SRC_URI += "git://git.ti.com/ti-linux-kernel/ti-linux-kernel.git;protocol=git; \
file://defconfig"
```

* - from meta-ti/recipes-kernel/linux/linux-ti-staging_4.14.bb

BOILING IT DOWN TO BASIC STEPS

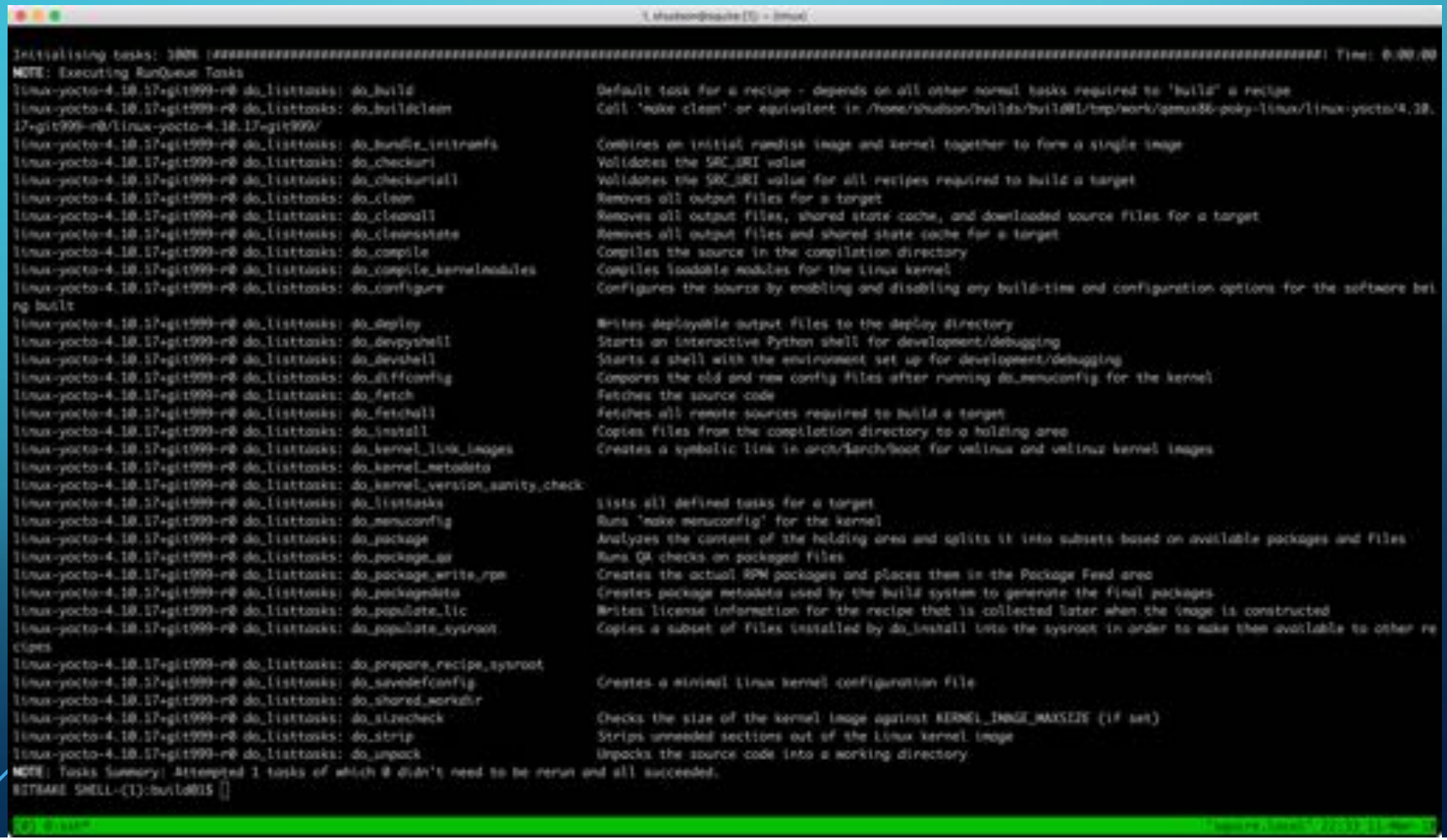
- Add the kernel source and configuration into the cross build (recipe)
- Verify your build
- Start making changes to the source and to the configuration
- Capture the changes

GENERAL: CAPTURING CHANGES

- How you capture changes influences recipe creation and vice-versa
- When capturing your changes you will need to decide where you want maintain them. As with all recipes, you can store the changes as a set of patches in the layer or as a revision in your VCS, which is usually git. This applies to both the kernel source itself, the kernel configuration, and the device tree source.
- Personally, I find that keeping more than a handful of patches in the layer becomes cumbersome and I prefer to keep as much as possible in git

IMPORTANT BITBAKE KERNEL TASKS

A QUICK SAMPLING OF BITBAKE KERNEL TASKS



IMPORTANT BITBAKE TASKS TO NOTE

- Bitbake provides a wrapper around the menuconfig target
  - e.g. bitbake –c menuconfig virtual/kernel
- Executes the kernel make target of the same name
- **Important Considerations:**
  - **This requires an existing configuration to start the process**
    - To get an initial .config, execute “bitbake –C kernel_configme virtual/kernel”
  - **This modifies the .config in the working source tree directly**
    - Make sure to capture these changes as subsequent fetches will overwrite your changes!

IMPORTANT BITBAKE TASKS TO NOTE (2)

- After executing menuconfig, which modifies the .config directly in the workdir, you should use the '-C' variant for the compile step. This tells Bitbake to invalidate the SSTATE cache and re-runs the task.
- bitbake virtual/kernel –C compile

IMPORTANT BITBAKE TASKS TO NOTE (3)

- Bitbake provides a wrapper around the diffconfig kernel script
- bitbake –c diffconfig (pay attention to where the output goes)
  - Wraps the kernel script of the same name
  - This generates a delta set of .config settings against the ‘baseline’
- Same considerations apply as to the ‘menuconfig’ task

CREATING/INTEGRATING CHANGES
AKA
BASIC DEVELOPER WORKFLOWS

STARTING FROM AN EXISTING KERNEL RECIPE

- Congratulations, you're most of the way home!
- Create a new layer with a .bbappend for your selected kernel
  - I suggest using 'yocto-layer create' to create a skeleton*
- If your kernel source and recipe support yocto-kernel tools, then create config fragments with the diffconfig command and incorporate them into your layer
- If your kernel does *not* support the yocto-kernel tools, then use menuconfig to create a new configuration and copy it to your layer
- Create patches for your kernel, add them to the layer, and add them to the recipe in the normal manner (more on this in a moment)

*** - As was pointed out during the live presentation, "yocto-layer" has been deprecated since Pyro.**

STARTING WITH A NEW KERNEL

- Go through the steps to add a kernel recipe to the build system (see simple example recipe above)
- Don't forget to add your new kernel to your local.conf so it get's used!
  - PREFERRED_PROVIDER_virtual/kernel ?= "linux-my-new-kernel"
- From there the steps are the same as on the previous slide

CREATING KERNEL PATCHES

- Multiple methods to choose from to create patches
  - Work in the workdir directly
    - Easiest to get to by using the "*bitbake virtual/kernel –c devshell*"
    - Be careful with this as you can lose work
  - Work with your source directly in another tree
    - You will not lose work accidentally from bitbake, but can be cumbersome
    - Since you're working in a VCS system…
      - You can generate patches, e.g. git format-patch
    - Syncing patches can be a pain
  - Devtool workflow

DEVTOOL

- A command-line tool that makes it easier to build, test, and package components in a Yocto Project build environment
- devtool creates a temporary workspace layer and adds it automatically into your bblayers.conf
- In that layer, you can work with live source, modify it, generate patches, if desired, and easily incorporate them back into your build layers
- It works well with the kernel and, once you have a recipe in place, you can easily setup a new workspace
  - 'devtool modify virtual/kernel' *
- Note: when you use 'devtool modify', it will automatically run the 'kernel_configme' task

CAPTURING KERNEL PATCHES

- The kernel source can in several forms: tarball, VCS (git), etc
- The form of the source chosen strongly influences the best way to capture patches
- General patterns
  - If your kernel recipe pulls a source tarball, it is generally easiest to capture patches directly into your layer
  - If your kernel recipe pulls from a VCS, create your patches directly in the VCS and update the recipe to track changes, as desired

SIMPLE, LIVE EXAMPLE

LET'S WORK THROUGH A SIMPLE

ADDITIONAL OBSERVATIONS/RECOMMENDATIONS

- Most developers start with a kernel provided by the semi or board manufacturer
- Take the path of least resistance!
  - For kernels that use the yocto-kernel tooling, use config fragments
  - For other kernels, use the basic template to keep things simple
- If you are going to maintain a kernel across multiple platforms, investing time to make your kernel work with the yocto-kernel tooling is worth it.
- Personally, I've found that working against a git repository and using devtool, I minimize the amount of overhead associated with regular builds.
- The tooling continues to improve with every release. So, keep checking to see what's changed!

QUESTIONS?

ADDITIONAL RESOURCES

- Yocto Project Developer's Day 2013 - Working with the Kernel
  - https://www.youtube.com/watch?v=rSaU091gcW8
- Customize your Mainline or LTSI Linux Kernel Using the Yocto Project
  - ELCE 2015
  - https://www.youtube.com/watch?v=ZwwPuJwcHNM
- Yocto Project Linux Kernel Development Manual
  - https://www.yoctoproject.org/docs/latest/kernel-dev/kernel-dev.html