Debian and Yocto Project: a Tale of Two Distros

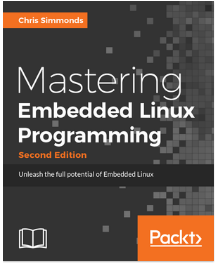
(one of which is not a distro)

Chris Simmonds

Embedded Linux Conference Europe 2020

License

About Chris Simmonds



- Consultant and trainer
- Author of *Mastering Embedded Linux Programming*
- Working with embedded Linux since 1999
- Android since 2009
- Speaker at many conferences and workshops

"Looking after the Inner Penguin" blog at `https://2net.co.uk/`

@2net_software

`https://uk.linkedin.com/in/chrisdsimmonds/`

Agenda

- Off the peg or bespoke?
- Board support
- Building the root file system
- Developing code
- Long term maintenance
- Conclusions

Choices

Off-the-peg Use a Debian-based distro

works out of the box

tens of thousands of packages

Choices

Off-the-peg Use a Debian-based distro

works out of the box

tens of thousands of packages

Bespoke Build everything from scratch using a build system like Yocto Project

tailored for your application

you are in control of everything

Debian

- Debian[1] is a binary distribution
  - so no need to cross-compile
- Popular on servers, desktops and embedded
  - also it is the basis of other distros, including Ubuntu
- Well known for stability, security and support

[1] You could equally well use other distros, including SUSE, Fedora, etc.

Yocto Project

- Yocto Project[1] is a build system that creates packages from source code
  - based on Bitbake and OpenEmbedded meta data
  - Yocto Project and OpenEmbedded have been used to create the software running on many millions of devices
- Allows you to create your own tailor-made distro
- You only need to build and deploy only the packages you need

[1] There are other build systems, for example Buildroot, that work in a similar way

Comparison

- A binary distro such as Debian is easy to use and set up ...
- ... but, you are restricted by the policy decisions of the distro
- Building your own distro with Yocto Project/OpenEmbedded is more effort ...
- ... but it gives you more control over the result

- Off the peg or bespoke?
- Board support
- Building the root file system
- Developing code
- Long term maintenance
- Conclusions

Which hardware is supported by Debian?

- The Debian project supports a range of CPU architectures
  https://www.debian.org/releases/stable/arm64/ch02s01.en.html

In Debian 9 (Stretch) and later, these are the architectures most relevant for embedded:

Architecture	Description
i386	x86
amd64	x86_64
armel	ARMv4T (32-bit)
armhf	ARMv7-A (32-bit) with floating point unit
arm64	ARMv8-A (64-bit)

Board Support for Debian 1/2

- As well as support for the architecture, you also need support for the specific board you are using
  - AKA Board Support Package, or BSP
- A BSP consists of
  - Bootloader (ARM) or BIOS (x86)
  - Kernel
  - Kernel drivers specific to the board
  - Device tree (ARM)
  - Libraries to support vendor-specific components such as accelerated graphics

Board Support for Debian 2/2

- For x86 and x86_64
  - There are many "embedded PC" boards that can run Debian "out of the box"
  - i.e. the BSP is just the BIOS that is flashed on the board when you receive it
- For ARM
  - Things are more complicated
  - In most cases you will need a full BSP from the vendor (and the vendor will be responsible for maintaining it)
  - For 64-bit ARM devices, you can most likely use the stock Debian kernel, but you still need to get the bootloader, device tree, additional kernel drivers and graphics libraries from the vendor

A special mention for the Raspberry Pi

- The Raspberry Pi is one of the most popular ARM based boards [1]
- raspberrypi.org creates and maintains Debian-based BSPs for all boards
- Raspbian (32-bit)
  - Raspbian is a Debian port compiled for ARM-v6 (32-bit)
- Raspberry PI OS (64-bit) - still in beta testing
  - Based on upstream Debian arm64

[1] Both in general computing and embedded

What about the Beagles?

- The BeagleBoard and BeagleBone boards from beagleboard.org are also very popular in embedded
- beagleboard.org provide Debian BSPs for their boards

Which hardware is supported by Yocto Project?

- In Yocto/OpenEmbedded, the BSP is usually provided by the board vendor as a meta layer
- Almost all vendors of ARM based boards do provide such a layer

Comparison

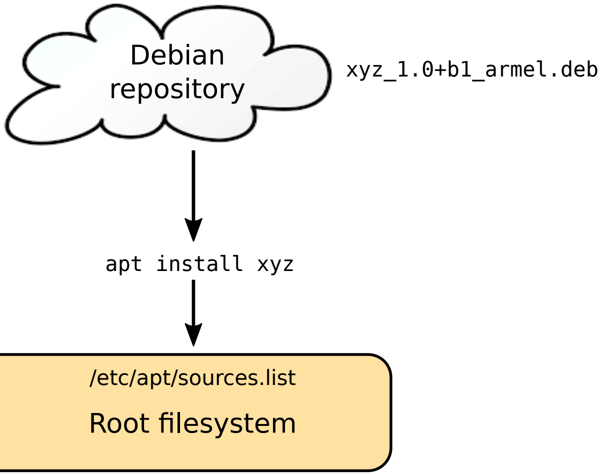
- The strength of Yocto Project/OpenEmbedded is the flexibility you have in defining the base hardware
- The strength of Debian is that for popular boards, you don't have to
- Creating or modifying a BSP is much easier (in my experience) with Yocto than with Debian

- Off the peg or bespoke?
- Board support
- Building the root file system
- Developing code
- Long term maintenance
- Conclusions

The root file system

- The root filesystem contains the programs, libraries, scripts, and configuration files needed to make the system run
- Usually created from a list of packages
- There are two methods
  - Start from scratch and create the list of packages you need
  - Start from a known working system and strip out the ones you don’t need

Building a Debian rootfs



The search for the "Golden Master"

The overall procedure would be

- Take a full desktop image
- Strip out things you don’t want
- Add the things you do want
- Add any other tweaks you need
- Once development is done, use dd (or similar) to take a copy of the filesystem to create a "Golden Master"
- Clone it to all units shipped

What can go wrong?

A Golden Master can become a millstone

- Steps to create it are almost certainly not documented

What can go wrong?

A Golden Master can become a millstone

- Steps to create it are almost certainly not documented
  - so changes have to be incremental
  - major changes, e.g. to a new distro release, are very difficult

What can go wrong?

A Golden Master can become a millstone

- Steps to create it are almost certainly not documented
    - so changes have to be incremental
    - major changes, e.g. to a new distro release, are very difficult
- Probably contains a finger-print of the person who created it

What can go wrong?

A Golden Master can become a millstone

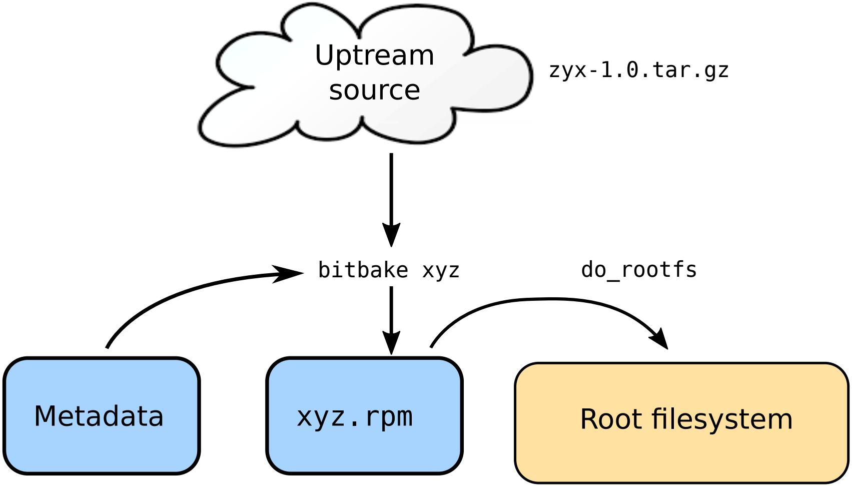
- Steps to create it are almost certainly not documented
  - so changes have to be incremental
  - major changes, e.g. to a new distro release, are very difficult
- Probably contains a finger-print of the person who created it
  - user accounts and passwords
  - `$HOME/.bash_history`
  - old system log files

Developing on Debian: second pass

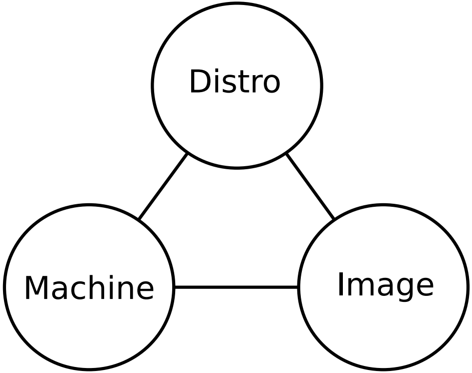
You need a **robust**, **reproducible** build process

- Install only the packages you need
- Import your own software and configuration (ideally encapsulated as Debian packages)
- Examples
  - ELBE https://elbe-rfs.org/
  - Armbian https://www.armbian.com/

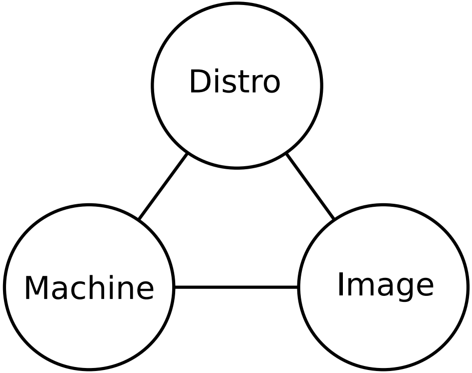
Building a rootfs with Yocto Project



It’s all in the metadata

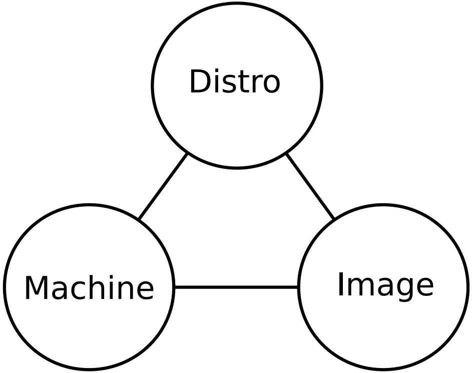


It’s all in the metadata



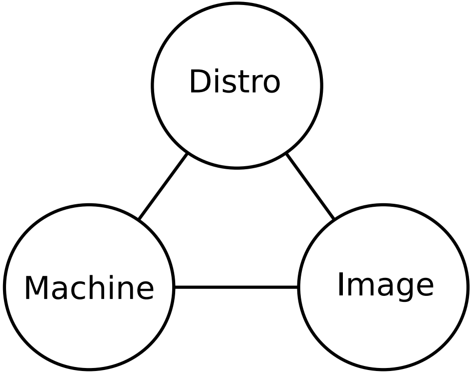
- **Distro**: how I want to put my system together

It’s all in the metadata



- **Distro**: how I want to put my system together
- **Machine**: the board I want to build for

It’s all in the metadata



- **Distro**: how I want to put my system together
- **Machine**: the board I want to build for
- **Image**: the selection of packages I want

Comparison

- Yocto Project/OpenEmbedded
  - Yocto, being source based, allows you much finer control over the configuration of packages
  - The packages are built with compile flags tuned to the exact CPU you are using
- Debian
  - The root file system will be bigger than with Yocto because don’t have any control over the contents of the packages
  - and, of course, please avoid creating hand-crafted golden masters

- Off the peg or bespoke?
- Board support
- Building the root file system
- Developing code
- Long term maintenance
- Conclusions

Compiling

- Assuming that you have developed some code written in C++ or another compiled language, how do you build it for the target?
- Cross compile
  - Compile on a fast desktop or server (probably running on x86_64)
  - Deploy on the target (probably running an ARM SoC)
- Native compile
  - Compile on the target

Developing code for Yocto

- Yocto Project uses cross compilers by default
- Yocto can create an SDK which contains
  - Cross compilers
  - Development packages for all the libraries used by the target
  - Debug tools
- Once you have installed the SDK, you can set up the environment and build on a (Linux) desktop

Developing code for Debian

- Debian uses native compilers by default
- So, you have to compile on the target, which creates problems because
    - the CPU is slow
    - you may run out of memory or storage
- Workaround: create a QEMU environment with the same architecture and libraries as your board, but more RAM and storage, and build there

Comparison

- In most cases, cross development is preferable
- So, Yocto is the clear winner

- Off the peg or bespoke?
- Board support
- Building the root file system
- Developing code
- Long term maintenance
- Conclusions

Long term support - Debian

- Releases are usually supported for at least 5 years (`https://wiki.debian.org/LTS`)
- Note that support for the BSP comes from the board vendor
  - if they don't use mainline kernel, this may be an issue

Long term support - Yocto Project/OpenEmbedded

- Yocto Project LTS releases are supported for 2 years
- As with Debian, support fro the BSP is the responsibilty of the board vendor

- Off the peg or bespoke?
- Board support
- Building the root file system
- Developing code
- Long term maintenance
- Conclusions

Conclusions

- It’s not either/or: both Debian and Yocto good choices
- Debian is best when
  - using commodity hardware with upstream kernel support
  - Proof Of Concept and prototypes
- Yocto is best when
  - you have custom hardware (no distro available)
  - optimized for minimal memory and storage

Any questions?